

Ant User Manual

by

- Stephane Bailliez (sbailliez@imediacion.com)
- Jacques Bergeron (jacques.bergeron@dogico.com)
- Stefan Bodewig (stefan.bodewig@epost.de)
- Patrick Chanezon (chanezon@netscape.com)
- James Duncan Davidson (duncan@x180.com)
- Tom Dimock (tad1@cornell.edu)
- Peter Donald (donaldp@apache.org)
- Diane Holt (holtdl@yahoo.com)
- Bill Kelly (bill.kelly@softwired-inc.com)
- Arnout J. Kuiper (ajkuiper@wx.nl)
- Conor MacNeill (conor@cortexebusiness.com.au)
- Stefano Mazzocchi (stefano@apache.org)
- Erik Meade (emeade@geekfarm.org)
- Sam Ruby (rubys@us.ibm.com)
- Nico Seessle (nico@seessle.de)
- Jon S. Stevens (jon@latchkey.com)
- Roger Vaughn (rvaughn@seaconinc.com)
- Dave Walend (dwalend@cs.tufts.edu)
- Phillip Wells (philwells@rocketmail.com)

Version: @VERSION@

\$Id: credits.html,v 1.5 2001/05/18 14:58:38 conor Exp \$

Copyright © 2000,2001 Apache Software Foundation. All rights Reserved.

Table of Contents

1	Introduction	1
2	Installing Ant	2
2.1	Getting Ant	2
2.2	System Requirements	2
2.3	Installing Ant	2
2.4	Building Ant	4
2.5	Library Dependencies	5
3	Running Ant	7
3.1	Running Ant by Hand	8
4	Using Ant	9
4.1	Writing a Simple Buildfile	9
5	Concepts and Types	17
5.1	Directory-based Tasks	17
5.2	build.sysclasspath	20
5.3	Description	21
5.4	PatternSet	22
5.5	FileList	24
5.6	FileSet	25
5.7	FilterSet	26
5.8	FilterSet	26
5.9	Filter	26
5.10	Filtersfile	26
5.11	Mapping File Names	28
5.12	Common Attributes of all Tasks	32
6	Core Tasks	33
6.1	Overview of Ant core tasks	33
6.2	Ant	37
6.3	AntCall	38
6.4	AntStructure	40
6.5	Apply/ExecOn	41
6.6	Available	45
6.7	Chmod	47
6.8	Condition	49
6.9	Copy	52
6.10	Copydir	54
6.11	Copyfile	56
6.12	Cvs	57
6.13	cvspass	59
6.14	Delete	60
6.15	Deltree	62
6.16	DependSet	63
6.17	Ear	65
6.18	Example	66
6.19	Echo	67

Table of Contents

6.20	Exec	68
6.21	Fail	70
6.22	Filter	71
6.23	FixCRLF	72
6.24	GenKey	76
6.25	Get	77
6.26	GUnzip	78
6.27	GZip	79
6.28	Jar	80
6.29	Java	83
6.30	Javac	85
6.31	Javadoc/Javadoc2	90
6.32	Mail	97
6.33	Mkdir	98
6.34	Move	99
6.35	Parallel	101
6.36	Patch	103
6.37	Pathconvert	104
6.38	Property	107
6.39	Record	110
6.40	Rename	112
6.41	Replace	113
6.42	Rmic	116
6.43	Sequential	119
6.44	SignJar	120
6.45	Sleep	121
6.46	Sql	122
6.47	Style	126
6.48	Tar	129
6.49	Taskdef	132
6.50	Touch	133
6.51	Tstamp	134
6.52	Typedef	136
6.53	Unjar/Unwar/Unzip	138
6.54	Untar	139
6.55	Unjar/Unwar/Unzip	140
6.56	Unjar/Unwar/Unzip	141
6.57	Uptodate	142
6.58	War	143
6.59	Zip	146
7	Optional Tasks	150
7.1	.NET classes for Ant	150
7.2	ANTLR	156
7.3	Cab	157
7.4	ClearCase Support	159
7.5	Continuous Support	162
7.6	Depend	165

7.7	Ant EJB Tasks User Manual	168
7.8	FTP	189
7.9	icontract	194
7.10	JavaCC	197
7.11	Javah	200
7.12	JDepend	202
7.13	JJTree	204
7.14	Jlink	206
7.15	JProbe	208
7.16	JUnit	215
7.17	JUnitReport	220
7.18	MMetrics	222
7.19	MAudit	224
7.20	MimeMail	226
7.21	MParse	228
7.22	Native2Ascii	229
7.23	NetRexxC	231
7.24	Perforce Tasks User Manual	234
7.25	Ant PropertyFile Task User Manual	241
7.26	Ant PvcS Task User Manual	244
7.27	RenameExtensions	248
7.28	Rpm	249
7.29	Script	250
7.30	Sound	252
7.31	Starteam	254
7.32	Stylebook	256
7.33	Telnet	257
7.34	Test	259
7.35	wljspc	260
7.36	XMLValidate	262
8	IDE Integration	263
8.1	About Antidote	263
8.2	AntRunner For JBuilder (unbundled)	265
8.3	AntWork Plugin for the Jext - Java Text Editor	265
8.4	NetBeans Forte for Java (unbundled)	266
8.5	Visual Age for Java Tasks and Plugin User Manual	266
9	Developing with Ant	276
9.1	Writing Your Own Task	276
9.2	Build Events	278
10	Ant API	279
11	License	279
12	Feedback	280

1 Introduction

Ant is a Java-based build tool. In theory, it is kind of like *make*, without *make*'s wrinkles.

1.0.1 Why?

Why another build tool when there is already *make*, *gnumake*, *nmake*, *jam*, and others? Because all those tools have limitations that Ant's original author couldn't live with when developing software across multiple platforms. Make-like tools are inherently shell-based: they evaluate a set of dependencies, then execute commands not unlike what you would issue on a shell. This means that you can easily extend these tools by using or writing any program for the OS that you are working on; however, this also means that you limit yourself to the OS, or at least the OS type, such as Unix, that you are working on.

Makefiles are inherently evil as well. Anybody who has worked on them for any time has run into the dreaded tab problem. "Is my command not executing because I have a space in front of my tab?!!" said the original author of Ant way too many times. Tools like Jam took care of this to a great degree, but still have yet another format to use and remember.

Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface.

Granted, this removes some of the expressive power that is inherent in being able to construct a shell command such as `'find . -name foo -exec rm {}'`, but it gives you the ability to be cross-platform - to work anywhere and everywhere. And hey, if you really need to execute a shell command, Ant has an `<exec>` task that allows different commands to be executed based on the OS it is executing on.

2 Installing Ant

2.1 Getting Ant

2.1.1 Binary Edition

The latest stable version of Ant can be downloaded from <http://jakarta.apache.org/builds/ant/release/v1.4/bin/>. If you like living on the edge, you can download the latest version from <http://jakarta.apache.org/builds/ant/nightly/>.

2.1.2 Source Edition

If you prefer the source edition, you can download the source for the latest Ant release from <http://jakarta.apache.org/builds/ant/release/v1.4/src/>. Again, if you prefer the edge, you can access the code as it is being developed via CVS. The Jakarta website has details on accessing CVS (<http://jakarta.apache.org/site/cvsindex.html>). Please checkout the jakarta-ant module. See the section Building Ant [p 4] on how to build Ant from the source code. You can also access the Ant CVS repository (<http://cvs.apache.org/viewcvs/jakarta-ant/>) on-line.

2.2 System Requirements

To build and use Ant, you must have a JAXP-compliant XML parser installed and available on your class-path.

The binary distribution of Ant includes the latest version of the Apache Crimson (<http://xml.apache.org/crimson/index.html>) XML parser. Please see <http://java.sun.com/xml/> for more information about JAXP. If you wish to use a different JAXP-compliant parser, you should remove `jaxp.jar` and `crimson.jar` from Ant's `lib` directory. You can then either put the jars from your preferred parser into Ant's `lib` directory or put the jars on the system classpath.

For the current version of Ant, you will also need a JDK installed on your system, version 1.1 or later. A future version of Ant will require JDK 1.2 or later.

2.3 Installing Ant

The binary distribution of Ant consists of three directories: `bin`, `docs` and `lib`. Only the `bin` and `lib` directories are required to run Ant. To install Ant, choose a directory and copy the distribution file there. This directory will be known as `ANT_HOME`.

Windows 95 and Windows 98 Note:

*On these systems, the script used to launch Ant will have problems if ANT_HOME is a long filename. This is due to limitations in the OS's handling of the "for" batch-file statement. It is recommended, therefore, that Ant be installed in a **short** path, such as C:\Ant.*

Before you can run ant there is some additional set up you will need to do:

- Add the bin directory to your path.
- Set the ANT_HOME environment variable to the directory where you installed Ant. On some operating systems the ant wrapper scripts can guess ANT_HOME (Unix dialects and Windows NT/2000) - but it is better to not rely on this behavior.
- Optionally, set the JAVA_HOME environment variable (see the Advanced [p 4] section below). This should be set to the directory where your JDK is installed.

Note: Do not install Ant's ant.jar file into the lib/ext directory of the JDK/JRE. Ant is an application, whilst the extension directory is intended for JDK extensions. In particular there are security restrictions on the classes which may be loaded by an extension.

2.3.1 Optional Tasks

Ant supports a number of optional tasks. An optional task is a task which typically requires an external library to function. The optional tasks are packaged separately from the core Ant tasks. This package is available in the same download directory as the core ant distribution. The current jar containing optional tasks is named jakarta-ant-1.4-optional.jar. This jar should be downloaded and placed in the lib directory of your Ant installation.

The external libraries required by each of the optional tasks is detailed in the Library Dependencies [p 5] section. These external libraries may either be placed in Ant's lib directory, where they will be picked up automatically, or made available on the system CLASSPATH environment variable.

2.3.2 Windows

Assume Ant is installed in c:\ant\. The following sets up the environment:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\jdk1.2.2
set PATH=%PATH%;%ANT_HOME%\bin
```

2.3.3 Unix (bash)

Assume Ant is installed in /usr/local/ant. The following sets up the environment:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/jdk-1.2.2
export PATH=${PATH}:${ANT_HOME}/bin
```

2.3.4 Advanced

There are lots of variants that can be used to run Ant. What you need is at least the following:

- The classpath for Ant must contain `ant.jar` and any jars/classes needed for your chosen JAXP-compliant XML parser.
 - When you need JDK functionality (such as for the `javac` [p 85] task or the `rmic` [p 116] task), then for JDK 1.1, the `classes.zip` file of the JDK must be added to the classpath; for JDK 1.2 or JDK 1.3, `tools.jar` must be added. The scripts supplied with Ant, in the `bin` directory, will add the required JDK classes automatically, if the `JAVA_HOME` environment variable is set.
 - When you are executing platform-specific applications, such as the `exec` [p 68] task or the `cvs` [p 57] task, the property `ant.home` must be set to the directory containing where you installed Ant. Again this is set by the Ant scripts to the value of the `ANT_HOME` environment variable.
-

2.4 Building Ant

To build Ant from source, you can either install the Ant source distribution or checkout the jakarta-ant module from CVS.

Once you have installed the source, change into the installation directory.

Set the `JAVA_HOME` environment variable to the directory where the JDK is installed. See Installing Ant [p 2] for examples on how to do this for your operating system.

Make sure you have downloaded any auxiliary jars required to build tasks you are interested in. These should either be available on the `CLASSPATH` or added to the `lib/optional` directory. See Library Dependencies [p 5] for a list of jar requirements for various features. Note that this will make the auxiliary jars available for the building of Ant only. For running Ant you will still need to make the jars available as described under Installing Ant [p 2] .

You are now ready to build Ant:

```
build -Ddist.dir=<directory_to_contain_Ant_distribution>
dist (Windows)
```

```
build.sh -Ddist.dir=<directory_to_contain_Ant_distribution>
dist (Unix)
```

This will create a binary distribution of Ant in the directory you specified.

The above action does the following:

- If necessary it will bootstrap the Ant code. Bootstrapping involves the manual compilation of enough Ant code to be able to run Ant. The bootstrapped Ant is used for the remainder of the build steps.
- Invokes the bootstrapped Ant with the parameters passed to the build script. In this case, these parameters define an Ant property value and specify the "dist" target in Ant's own `build.xml` file.

On most occasions you will not need to explicitly bootstrap Ant since the build scripts do that for you. If however, the build file you are using makes use of features not yet compiled into the bootstrapped Ant, you will need to manually bootstrap. Run `bootstrap.bat` (Windows) or `bootstrap.sh` (UNIX) to build a new bootstrap version of Ant.

If you wish to install the build into the current `ANT_HOME` directory, you can use:

```
build install (Windows)
```

```
build.sh install (Unix)
```

You can avoid the lengthy Javadoc step, if desired, with:

```
build install-lite (Windows)
```

```
build.sh install-lite (Unix)
```

This will only install the `bin` and `lib` directories.

Both the `install` and `install-lite` targets will overwrite the current Ant version in `ANT_HOME`.

2.5 Library Dependencies

The following libraries are needed in your `CLASSPATH` or in the install directory's `lib` directory if you are using the indicated feature. Note that only one of the `regexp` libraries is needed for use with the mappers. You will also need to install the Ant optional jar containing the task definitions to make these tasks available. Please refer to the Installing Ant / Optional Tasks [p 3] section above.

Jar Name	Needed For	Available At
An XSL transformer like Xalan or XSL:P	style task	http://xml.apache.org/xalan-j/index.html or http://www.clc-marketing.com/xslp/
jakarta-regexp-1.2.jar	regexp type with mappers	jakarta.apache.org/regexp/
jakarta-oro-2.0.1.jar	regexp type with mappers and the performe tasks	jakarta.apache.org/oro/
junit.jar	junit tasks	www.junit.org
stylebook.jar	stylebook task	CVS repository of xml.apache.org
testlet.jar	test task	java.apache.org/framework
antlr.jar	antlr task	www.antlr.org
bsf.jar	script task	oss.software.ibm.com/developerworks/projects/bsf
netrexx.jar	netrexx task	www2.hursley.ibm.com/netrexx
rhino.jar	javascript with script task	www.mozilla.org
jpython.jar	python with script task	www.jpython.org
netcomponents.jar	ftp and telnet tasks	www.savarese.org/oro/downloads

3 Running Ant

Running Ant is simple, when you installed it as described in the previous section. Just type `ant`.

When nothing is specified, Ant looks for a `build.xml` file in the current directory. If found, it uses that file as the buildfile. If you use the `-find` option, Ant will search for a buildfile in the parent directory, and so on, until the root of the filesystem has been reached. To make Ant use another buildfile, use the command-line option `-buildfile file`, where *file* is the buildfile you want to use.

You can also set properties that override properties specified in the buildfile (see the property [p 107] task). This can be done with the `-Dproperty=value` option, where *property* is the name of the property, and *value* is the value for that property. This can also be used to pass in the value of some environment variables. You can also access environment variables using the property [p 107] task. Just pass `-DMYVAR=%MYVAR%` (Windows) or `-DMYVAR=$MYVAR` (Unix) to Ant - you can then access these variables inside your buildfile as `${MYVAR}`.

Two more options are: `-quiet`, which instructs Ant to print less information on the console when running, and `-verbose`, which causes Ant to print additional information to the console.

It is also possible to specify one or more targets that should be executed. When omitted, the target that is specified in the `default` attribute of the `<project>` tag is used.

The `-projecthelp` option prints out the description of the project, if it exists, followed by a list of this project's targets. First those with a description, then those without one.

Command-line option summary:

```
ant [options] [target [target2 [target3] ...]]
Options:
-help                print this message
-projecthelp         print project help information
-version             print the version information and exit
-quiet              be extra quiet
-verbose            be extra verbose
-debug              print debugging information
-emacs              produce logging information without adornments
-logfile file        use given file for log output
-logger classname   the class that is to perform logging
-listener classname add an instance of class as a project listener
-buildfile file      use specified buildfile
-find file           search for buildfile towards the root of the filesystem and use the first one found
-Dproperty=value     set property to value
```

3.0.1 Examples

```
ant
```

runs Ant using the `build.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml
```

runs Ant using the `test.xml` file in the current directory, on the default target.

```
ant -buildfile test.xml dist
```

runs Ant using the `test.xml` file in the current directory, on a target called `dist`.

```
ant -buildfile test.xml -Dbuild=build/classes dist
```

runs Ant using the `test.xml` file in the current directory, on a target called `dist`, setting the `build` property to the value `build/classes`.

3.0.2 Files

The Ant wrapper script for Unix will source (read and evaluate) the file `~/.antrc` before it does anything - the Windows batch file invokes `%HOME%\antrc_pre.bat` at the start and `%HOME%\antrc_post.bat` at the end. You can use these files to set/unset environment variables that should only be visible during the execution of Ant. See the next section for example.

3.0.3 Environment Variables

The wrapper scripts use the following environment variables (if set):

- `JAVACMD` full path of the Java executable. Use this to invoke a different JVM than `JAVA_HOME/bin/java(.exe)`.
- `ANT_OPTS` command line arguments that should be passed to the JVM - for example, you can define properties or set the maximum Java heap size here.

3.1 Running Ant by Hand

If you have installed Ant in the do-it-yourself way, Ant can be started with:

```
java -Dant.home=c:\ant org.apache.tools.ant.Main [options] [target]
```

These instructions actually do exactly the same as the `ant` command. The options and target are the same as when running Ant with the `ant` command. This example assumes you have set your classpath to include:

- `ant.jar`
- jars/classes for your XML parser
- the JDK's required jar/zip files

4 Using Ant

4.1 Writing a Simple Buildfile

Ant's buildfiles are written in XML. Each buildfile contains one project.

Each task element of the buildfile can have an `id` attribute and can later be referred to by the value supplied to this. The value has to be unique. (For additional information, see the Tasks [p 10] section below.)

4.1.1 Projects

A *project* has three attributes:

Attribute	Description	Required
name	the name of the project.	No
default	the default target to use when no target is supplied.	Yes
basedir	the base directory from which all path calculations are done. This attribute might be overridden by setting the "basedir" property beforehand. When this is done, it must be omitted in the project tag. If neither the attribute nor the property have been set, the parent directory of the buildfile will be used.	No

Optionally, a description for the project can be provided as a top-level `<description>` element (see the description [p 21] type).

Each project defines one or more *targets*. A target is a set of *tasks* you want to be executed. When starting Ant, you can select which target(s) you want to have executed. When no target is given, the project's default is used.

4.1.2 Targets

A target can depend on other targets. You might have a target for compiling, for example, and a target for creating a distributable. You can only build a distributable when you have compiled first, so the distribute target *depends on* the compile target. Ant resolves these dependencies.

It should be noted, however, that Ant's `depends` attribute only specifies the *order* in which targets should be executed - it does not affect whether the target that specifies the dependency(s) gets executed if the dependent target(s) did not (need to) run.

Ant tries to execute the targets in the `depends` attribute in the order they appear (from left to right). Keep in mind that it is possible that a target can get executed earlier when an earlier target depends on it:

```

<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>

```

Suppose we want to execute target D. From its `depends` attribute, you might think that first target C, then B and then A is executed. Wrong! C depends on B, and B depends on A, so first A is executed, then B, then C, and finally D.

A target gets executed only once, even when more than one target depends on it (see the previous example).

A target also has the ability to perform its execution if (or unless) a property has been set. This allows, for example, better control on the building process depending on the state of the system (java version, OS, command-line property defines, etc.). To make a target *sense* this property, you should add the `if` (or `unless`) attribute with the name of the property that the target should react to. For example:

```

<target name="build-module-A" if="module-A-present"/>

<target name="build-own-fake-module-A" unless="module-A-present"/>

```

If no `if` and no `unless` attribute is present, the target will always be executed.

The optional `description` attribute can be used to provide a one-line description of this target, which is printed by the `-projecthelp` command-line option.

It is a good practice to place your `tstamp` [p 134] tasks in a so-called *initialization* target, on which all other targets depend. Make sure that target is always the first one in the `depends` list of the other targets. In this manual, most initialization targets have the name `"init"`.

A target has the following attributes:

Attribute	Description	Required
name	the name of the target.	Yes
depends	a comma-separated list of names of targets on which this target depends.	No
if	the name of the property that must be set in order for this target to execute.	No
unless	the name of the property that must not be set in order for this target to execute.	No
description	a short description of this target's function.	No

4.1.3 Tasks

A task is a piece of code that can be executed.

A task can have multiple attributes (or arguments, if you prefer). The value of an attribute might contain references to a property. These references will be resolved before the task is executed.

Tasks have a common structure:

```
<name attribute1="value1" attribute2="value2" ... />
```

where *name* is the name of the task, *attributeN* is the attribute name, and *valueN* is the value for this attribute.

There is a set of built-in tasks [p 17] , along with a number of optional tasks [p 150] , but it is also very easy to write your own [p 276] .

All tasks share a task name attribute. The value of this attribute will be used in the logging messages generated by Ant.

Tasks can be assigned an *id* attribute:

```
<taskname id="taskID" ... />
```

where *taskname* is the name of the task, and *taskID* is a unique name for this task. You can refer to the corresponding task object in scripts or other tasks via this name. For example, in scripts you could do:

```
<script ... >  
    task1.setFoo("bar");  
</script>
```

to set the *foo* attribute of this particular task instance. In another task (written in Java), you can access the instance via `project.getReference("task1")`.

Note¹: If "task1" has not been run yet, then it has not been configured (ie., no attributes have been set), and if it is going to be configured later, anything you've done to the instance may be overwritten.

Note²: Future versions of Ant will most likely *not* be backward-compatible with this behaviour, since there will likely be no task instances at all, only proxies.

4.1.4 Properties

A project can have a set of properties. These might be set in the buildfile by the property task [p 107] , or might be set outside Ant. A property has a name and a value. Properties may be used in the value of task attributes. This is done by placing the property name between "\${" and "}" in the attribute value. For example, if there is a "builddir" property with the value "build", then this could be used in an attribute like this: `${builddir}/classes`. This is resolved as `build/classes`.

4.1.5 Built-in Properties

Ant provides access to all system properties as if they had been defined using a `<property>` task. For example, `${OS.name}` expands to the name of the operating system.

For a list of system properties see the Javadoc of `System.getProperties` ([http://java.sun.com/j2se/1.3/docs/api/java/lang/System.html#getProperties\(\)](http://java.sun.com/j2se/1.3/docs/api/java/lang/System.html#getProperties())).

In addition, Ant has some built-in properties:

basedir	the absolute path of the project's basedir (as set with the basedir attribute of <project>).
ant.file	the absolute path of the buildfile.
ant.version	the version of Ant
ant.project.name	the name of the project that is currently executing; it is set in the name attribute of <project>.
ant.java.version	the JVM version Ant detected; currently it can hold the values "1.1", "1.2", "1.3" and "1.4".

4.1.6 Example

```
<project name="MyProject" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="src" value="."/>
  <property name="build" value="build"/>
  <property name="dist" value="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

4.1.7 Token Filters

A project can have a set of tokens that might be automatically expanded if found when a file is copied, when the filtering-copy behavior is selected in the tasks that support this. These might be set in the build-file by the filter task [p 71] .

Since this can potentially be a very harmful behavior, the tokens in the files **must** be of the form `@token@`, where *token* is the token name that is set in the filter task. This token syntax matches the syntax of other build systems that perform such filtering and remains sufficiently orthogonal to most programming and scripting languages, as well as with documentation systems.

Note: If a token with the format `@token@` is found in a file, but no filter is associated with that token, no changes take place; therefore, no escaping method is available - but as long as you choose appropriate names for your tokens, this should not cause problems.

Warning: If you copy binary files with filtering turned on, you can corrupt the files. This feature should be used with text files *only*.

4.1.8 Path-like Structures

You can specify PATH- and CLASSPATH-type references using both ":" and ";" as separator characters. Ant will convert the separator to the correct character of the current operating system.

Wherever path-like values need to be specified, a nested element can be used. This takes the general form of:

```
<classpath>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

The `location` attribute specifies a single file or directory relative to the project's base directory (or an absolute filename), while the `path` attribute accepts colon- or semicolon-separated lists of locations. The `path` attribute is intended to be used with predefined paths - in any other case, multiple elements with `location` attributes should be preferred.

As a shortcut, the `<classpath>` tag supports `path` and `location` attributes of its own, so:

```
<classpath>
  <pathelement path="${classpath}"/>
</classpath>
```

can be abbreviated to:

```
<classpath path="${classpath}"/>
```

In addition, FileSet [p 25] s can be specified via nested `<fileset>` elements. The order in which the files building up a fileset are added to the path-like structure is not defined.

```
<classpath>
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
</classpath>
```

This builds a path that holds the value of `${classpath}`, followed by all jar files in the `lib` directory, followed by the `classes` directory.

If you want to use the same path-like structure for several tasks, you can define them with a `<path>` element at the same level as *targets*, and reference them via their *id* attribute - see References [p 15] for an example.

A path-like structure can include a reference to another path-like structure via nested `<path>` elements:

```
<path id="base.path">
  <pathelement path="${classpath}"/>
  <fileset dir="lib">
    <include name="**/*.jar"/>
  </fileset>
  <pathelement location="classes"/>
</path>

<path id="tests.path">
  <path refid="base.path"/>
  <pathelement location="testclasses"/>
</path>
```

The shortcuts previously mentioned for `<classpath>` are also valid for `<path>`. For example:

```
<path id="base.path">
  <pathelement path="${classpath}"/>
</path>
```

can be written as:

```
<path id="base.path" path="${classpath}"/>
```

4.1.9 Command-line Arguments

Several tasks take arguments that will be passed to another process on the command line. To make it easier to specify arguments that contain space characters, nested `arg` elements can be used.

Attribute	Description	Required
value	a single command-line argument; can contain space characters.	Exactly one of these.
line	a space-delimited list of command-line arguments.	
file	The name of a file as a single command-line argument; will be replaced with the absolute filename of the file.	
path	A string that will be treated as a path-like string as a single command-line argument; you can use <code>;</code> or <code>:</code> as path separators and Ant will convert it to the platform's local conventions.	

4.1.9.1 Examples

```
<arg value="-l -a"/>
```

is a single command-line argument containing a space character.

```
<arg line="-l -a"/>
```

represents two separate command-line arguments.

```
<arg path="/dir;/dir2;\dir3"/>
```

is a single command-line argument with the value `\dir;\dir2;\dir3` on DOS-based systems and `/dir:/dir2:/dir3` on Unix-like systems.

4.1.10 References

The `id` attribute of the buildfile's elements can be used to refer to them. This can be useful if you are going to replicate the same snippet of XML over and over again - using a `<classpath>` structure more than once for example.

The following example:

```
<project ... >
  <target ... >
    <rmic ...>
      <classpath>
        <pathelement location="lib"/>
        <pathelement path="${java.class.path}"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </rmic>
  </target>

  <target ... >
    <javac ...>
      <classpath>
        <pathelement location="lib"/>
        <pathelement path="${java.class.path}"/>
        <pathelement path="${additional.path}"/>
      </classpath>
    </javac>
  </target>
</project>
```

could be rewritten as:

```
<project ... >
  <path id="project.class.path">
    <pathelement location="lib"/>
    <pathelement path="${java.class.path}"/>
    <pathelement path="${additional.path}"/>
  </path>
```

```
<target ... >
  <rmic ...>
    <classpath refid="project.class.path"/>
  </rmic>
</target>

<target ... >
  <javac ...>
    <classpath refid="project.class.path"/>
  </javac>
</target>
</project>
```

All tasks that use nested elements for `PatternSet` [p 22] s, `FileSet` [p 25] s or path-like structures [p 13] accept references to these structures as well.

5 Concepts and Types

5.1 Directory-based Tasks

Some tasks use directory trees for the task they perform. For example, the `javac` [p 85] task, which works upon a directory tree with `.java` files. Sometimes it can be very useful to work on a subset of that directory tree. This section describes how you can select a subset of such a directory tree.

Ant gives you two ways to create a subset, both of which can be used at the same time:

- Only include files/directories that match at least one pattern of a set of patterns.
- Exclude files/directories that match at least one pattern a set of patterns.

When both inclusion and exclusion are used, only files/directories that match the include patterns, and don't match the exclude patterns, are used.

Patterns can be specified inside the buildfile via task attributes or nested elements and via external files. Each line of the external file is taken as a pattern that is added to the list of include or exclude patterns.

5.1.1 Patterns

As described earlier, patterns are used for the inclusion and exclusion. These patterns look very much like the patterns used in DOS and UNIX:

'*' matches zero or more characters, '?' matches one character.

Examples:

`*.java` matches `.java`, `x.java` and `FooBar.java`, but not `FooBar.xml` (does not end with `.java`).

`?java` matches `x.java`, `A.java`, but not `.java` or `xyz.java` (both don't have one character before `.java`).

Combinations of '*'s and '?'s are allowed.

Matching is done per-directory. This means that first the first directory in the pattern is matched against the first directory in the path to match. Then the second directory is matched, and so on. For example, when we have the pattern `/?abc/*/*.java` and the path `/xabc/foobar/test.java`, the first `?abc` is matched with `xabc`, then `*` is matched with `foobar`, and finally `*.java` is matched with `test.java`. They all match, so the path matches the pattern.

To make things a bit more flexible, we add one extra feature, which makes it possible to match multiple directory levels. This can be used to match a complete directory tree, or a file anywhere in the directory tree. To do this, `**` must be used as the name of a directory. When `**` is used as the name of a directory in the pattern, it matches zero or more directories. For example: `/test/**` matches all files/directories under `/test/`, such as `/test/x.java`, or `/test/foo/bar/xyz.html`, but not `/xyz.xml`.

There is one "shorthand" - if a pattern ends with / or \, then ** is appended. For example, `mypackage/test/` is interpreted as if it were `mypackage/test/**`.

Example patterns:

<code>**/CVS/*</code>	Matches all files in CVS directories that can be located anywhere in the directory tree. Matches: <pre>CVS/Repository org/apache/CVS/Entries org/apache/jakarta/tools/ant/CVS/Entries</pre> But not: <pre>org/apache/CVS/foo/bar/Entries (foo/bar/ part does not match)</pre>
<code>org/apache/jakarta/**</code>	Matches all files in the <code>org/apache/jakarta</code> directory tree. Matches: <pre>org/apache/jakarta/tools/ant/docs/index.html org/apache/jakarta/test.xml</pre> But not: <pre>org/apache/xyz.java (jakarta/ part is missing).</pre>
<code>org/apache/**/CVS/*</code>	Matches all files in CVS directories that are located anywhere in the directory tree under <code>org/apache</code>. Matches: <pre>org/apache/CVS/Entries org/apache/jakarta/tools/ant/CVS/Entries</pre> But not: <pre>org/apache/CVS/foo/bar/Entries (foo/bar/ part does not match)</pre>
<code>**/test/**</code>	Matches all files that have a <code>test</code> element in their path, including <code>test</code> as a filename.

When these patterns are used in inclusion and exclusion, you have a powerful way to select just the files you want.

5.1.2 Examples

```
<copy todir="${dist}">
  <fileset dir="${src}"
    includes="**/images/*"
    excludes="**/*.gif"
  />
</copy>
```

This copies all files in directories called `images` that are located in the directory tree defined by `${src}` to the destination directory defined by `${dist}`, but excludes all `*.gif` files from the copy.

This example can also be expressed using nested elements:

```
<copy todir="${dist}">
  <fileset dir="${src}">
    <include name="**/images/**"/>
    <exclude name="**/*.gif"/>
  </fileset>
</copy>
```

5.1.3 Default Excludes

There are a set of definitions that are excluded by default from all directory-based tasks. They are:

```
**/*~
**/#*#
**/.#*
**/%*%
**/CVS
**/CVS/**
**/.cvsignore
**/SCCS
**/SCCS/**
**/vssver.scc
```

If you do not want these default excludes applied, you may disable them with the `defaultexcludes="no"` attribute.

5.2 build.sysclasspath

The value of the build.sysclasspath property control how the system classpath, ie. the classpath in effect when Ant is run, affects the behaviour of classpaths in Ant. The default behavior varies from Ant to Ant task.

The values and their meanings are:

- only** Only the system classpath is used and classpaths specified in build files, etc are ignored. This situation could be considered as the person running the build file knows more about the environment than the person writing the build file
- ignore** The system classpath is ignored. This situation is the reverse of the above. The person running the build trusts the build file writer to get the build file right
- last** The classpath is concatenated to any specified classpaths at the end. This is a compromise, where the build file writer has priority.
- first** Any specified classpaths are concatenated to the system classpath. This is the other form of compromise where the build runner has priority.

5.3 Description

5.3.1 *Description*

Provides a description of the current project.

5.3.2 *Parameters*

(none)

5.3.3 *Examples*

```
<description>  
This buildfile is used to build the foo subproject within  
the large, complex bar project.  
</description>
```

5.4 PatternSet

Patterns [p 17] can be grouped to sets and later be referenced by their `id` attribute. They are defined via a `patternset` element, which can appear nested into a `FileSet` [p 25] or a directory-based task that constitutes an implicit `FileSet`. In addition, `patternsets` can be defined as a stand alone element at the same level as `target` — i.e., as children of `project` as well as as children of `target`.

Patterns can be specified by nested `<include>`, or `<exclude>` elements or the following attributes.

Attribute	Description
<code>includes</code>	comma-separated list of patterns of files that must be included. All files are included when omitted.
<code>includesfile</code>	the name of a file; each line of this file is taken to be an include pattern. You can specify more than one include file by using a nested <code>includesfile</code> elements.
<code>excludes</code>	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.
<code>excludesfile</code>	the name of a file; each line of this file is taken to be an exclude pattern. You can specify more than one exclude file by using a nested <code>excludesfile</code> elements.

5.4.1 Parameters specified as nested elements

5.4.1.1 `include` and `exclude`

Each such element defines a single pattern for files to include or exclude.

Attribute	Description	Required
<code>name</code>	the pattern [p 17] to in/exclude.	Yes
<code>if</code>	Only use this pattern if the named property is set.	No
<code>unless</code>	Only use this pattern if the named property is not set.	No

5.4.1.2 `includesfile` and `excludesfile`

If you want to list the files to include or exclude external to your build file, you should use the `includesfile/excludesfile` attributes or elements. Using the attribute, you can only specify a single file of each type, while the nested elements can be specified more than once - the nested elements also support `if/unless` attributes you can use to test the existence of a property.

Attribute	Description	Required
<code>name</code>	the name of the file holding the patterns to in/exclude.	Yes
<code>if</code>	Only read this file if the named property is set.	No
<code>unless</code>	Only read this file if the named property is not set.	No

5.4.2 Examples

```
<patternset id="non.test.sources" >
  <include name="**/*.java"/>
  <exclude name="**/*Test*"/>
</patternset>
```

Builds a set of patterns that matches all `.java` files that do not contain the text `Test` in their name. This set can be referred [p 15] to via `<patternset refid="non.test.sources"/>`, by tasks that support this feature, or by `FileSets`.

Note that while the `includes` and `excludes` attributes accept multiple elements separated by commas or spaces, the nested `<include>` and `<exclude>` elements expect their `name` attribute to hold a single pattern.

The nested elements allow you to use `if` and `unless` arguments to specify that the element should only be used if a property is set, or that it should be used only if a property is not set.

For example

```
<patternset id="sources" >
  <include name="std/**/*.java"/>
  <include name="prof/**/*.java" if="professional"/>
  <exclude name="**/*Test*"/>
</patternset>
```

will only include the files in the sub-directory *prof* if the property *professional* is set to some value.

The two sets

```
<patternset includesfile="some-file" />
```

and

```
<patternset>
  <includesfile name="some-file" />
</patternset/>
```

are identical. The include patterns will be read from the file `some-file`, one pattern per line.

```
<patternset>
  <includesfile name="some-file" />
  <includesfile name="${some-other-file}"
                if="some-other-file"
  />
</patternset/>
```

will also read include patterns from the file the property `some-other-file` points to, if a property of that name has been defined.

5.5 FileList

FileLists are explicitly named lists of files. Whereas FileSets act as filters, returning only those files that exist in the file system and match specified patterns, filelists are useful for specifying filenames that may or may not exist. Multiple files are specified via a comma-separated list, with no support for wildcards. FileLists can appear inside tasks that support this feature or at the same level as `target` - i.e., as children of `project`.

Attribute	Description	Required
dir	the base directory of this FileList.	Yes
files	Comma-separated list of file names.	Yes

5.5.0.1 Examples

```
<filelist
  id="docfiles"
  dir="${doc.src}"
  files="foo.xml,bar.xml"/>
```

The files `${doc.src}/foo.xml` and `${doc.src}/bar.xml`. Note that these files may not (yet) actually exist.

```
<filelist refid="docfiles"/>
```

Same files as the example above.

5.6 FileSet

FileSets are groups of files. These files can be found in a directory tree starting in a base directory and are matched by patterns taken from a number of PatternSets [p 22] . FileSets can appear inside tasks that support this feature or at the same level as `target` - i.e., as children of `project`.

PatternSets can be specified as nested `<patternset>` elements. In addition, FileSet holds an implicit PatternSet and supports the nested `<include>`, `<includesfile>`, `<exclude>` and `<excludesfile>` elements of PatternSet directly, as well as PatternSet's attributes.

Attribute	Description	Required
dir	the root of the directory tree of this FileSet.	Yes
defaultexcludes	indicates whether default excludes should be used or not (yes no); default excludes are used when omitted.	No
includes	comma-separated list of patterns of files that must be included; all files are included when omitted.	No
includesfile	the name of a file; each line of this file is taken to be an include pattern.	No
excludes	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file; each line of this file is taken to be an exclude pattern.	No

5.6.0.1 Examples

```
<fileset dir="${server.src}" >
  <patternset id="non.test.sources" >
    <include name="**/*.java"/>
    <exclude name="**/*Test*" />
  </patternset>
</fileset>
```

Groups all files in directory `${server.src}` that are Java source files and don't have the text `Test` in their name.

```
<fileset dir="${client.src}" >
  <patternset refid="non.test.sources"/>
</fileset>
```

Groups all files in directory `${client.src}`, using the same patterns as the example above.

5.7 Filterset

FilterSets are groups of filters. Filters can be defined as token value pairs or be read in from a file. FilterSets can appear inside tasks that support this feature or at the same level as **target** - i.e., as children of **project**.

In addition, Filtersets can have begintoken and/or endtoken attributes to define what to match. Filtersets are used for doing replacements in tasks like copy etc.

5.8 Filterset

Attribute	Description	Default	Required
starttoken	The string marking the beginning of a token. eg @Date@	@	No
endtoken	The string marking the end of a token. eg @Date@	@	No

5.9 Filter

Attribute	Description	Required
token	The token to replace eg @Date@	Yes
value	The value to replace it with eg Thursday, April 26, 2001	Yes

5.10 Filtersfile

Attribute	Description	Required
file	The file to load tokens from should be a properties file of name value pairs.	Yes

5.10.0.1 Examples

You are copying the version.txt file to the dist directory from the build directory but wish to replace the token @DATE@ with todays date.

```
<copy file="${build.home}/version.txt" toFile="${dist.home}/version.txt">
  <filterset>
    <filter token="DATE" value="${DATE}"/>
  </filterset>
</copy>
```

You are copying the version.txt file to the dist directory from the build directory but wish to replace the token %DATE* with todays date.

```
<copy file="${build.home}/version.txt" toFile="${dist.home}/version.txt">
  <filterset begintoken="%" endtoken="*">
    <filter token="DATE" value="${DATE}"/>
  </filterset>
</copy>
```

Copy all the docs but change all dates and appropriate notices as stored in a file.

```
<copy toDir="${dist.home}/docs" >
  <fileset dir="${build.home}/docs">
    <include name="**/*.html">
  </fileset>
  <filterset begintoken="%" endtoken="%">
    <filter token="DATE" value="${DATE}"/>
    <filtersfile file="${user.home}/dist.properties"/>
  </filterset>
</copy>
```

5.11 Mapping File Names

Some tasks take source files and create target files. Depending on the task, it may be quite obvious which name a target file will have (using `javac` [p 85], you know there will be `.class` files for your `.java` files) - in other cases you may want to specify the target files, either to help Ant or to get an extra bit of functionality.

While source files are usually specified as `fileset` [p 25] s, you don't specify target files directly - instead, you tell Ant how to find the target file(s) for one source file. An instance of `org.apache.tools.ant.util.FileNameMapper` is responsible for this. It constructs target file names based on rules that can be parameterized with `from` and `to` attributes - the exact meaning of which is implementation-dependent.

These instances are defined in `<mapper>` elements with the following attributes:

Attribute	Description	Required
type	specifies one of the built-in implementations.	Exactly one of both
classname	specifies the implementation by class name.	
classpath	the classpath to use when looking up <code>classname</code> .	No
classpathref	the classpath to use, given as reference [p 15] to a path defined elsewhere.	No
from	the <code>from</code> attribute for the given implementation.	Depends on implementation.
to	the <code>to</code> attribute for the given implementation.	Depends on implementation.

Note that Ant will not automatically convert `/` or `\` characters in the `to` and `from` attributes to the correct directory separator of your current platform. If you need to specify this separator, use `${file.separator}` instead.

The classpath can be specified via a nested `<classpath>`, as well - that is, a path [p 13] -like structure.

The built-in mapper types are:

5.11.0.1 identity

The target file name is identical to the source file name. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="identity"/>
```

Source file name	Target file name
A.java	A.java
foo/bar/B.java	foo/bar/B.java
C.properties	C.properties
Classes/dir/dir2/A.properties	Classes/dir/dir2/A.properties

5.11.0.2 flatten

The target file name is identical to the source file name, with all leading directory information stripped off. Both `to` and `from` will be ignored.

Examples:

```
<mapper type="flatten"/>
```

Source file name	Target file name
A.java	A.java
foo/bar/B.java	B.java
C.properties	C.properties
Classes/dir/dir2/A.properties	A.properties

5.11.0.3 merge

The target file name will always be the same, as defined by `to` - `from` will be ignored.

5.11.0.3.1 Examples:

```
<mapper type="merge" to="archive.tar"/>
```

Source file name	Target file name
A.java	archive.tar
foo/bar/B.java	archive.tar
C.properties	archive.tar
Classes/dir/dir2/A.properties	archive.tar

5.11.0.4 glob

Both `to` and `from` define patterns that may contain at most one `*`. For each source file that matches the `from` pattern, a target file name will be constructed from the `to` pattern by substituting the `*` in the `to` pattern with the text that matches the `*` in the `from` pattern. Source file names that don't match the `from` pattern will be ignored.

Examples:

```
<mapper type="glob" from="*.java" to="*.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak
C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="glob" from="C*ies" to="Q*y"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	ignored
C.properties	Q.property
Classes/dir/dir2/A.properties	Qlasses/dir/dir2/A.property

5.11.0.5 regexp

Both `to` and `from` define regular expressions. If the source file name matches the `from` pattern, the target file name will be constructed from the `to` pattern, using `\0` to `\9` as back-references for the full match (`\0`) or the matches of the subexpressions in parentheses. Source files not matching the `from` pattern will be ignored.

Note that you need to escape a dollar-sign (\$) with another dollar-sign in Ant.

The regexp mapper needs a supporting library and an implementation of `org.apache.tools.ant.util.regexp.RegexpMatcher` that hides the specifics of the library. Ant comes with implementations for the `java.util.regex` package of JDK 1.4 (<http://java.sun.com/j2se/1.4/docs/api/java/util/regex/package-summary.html>), jakarta-regexp (<http://jakarta.apache.org/regexp/>) and jakarta-ORO (<http://jakarta.apache.org/oro/>). If you compile from sources and plan to use one of them, make sure the libraries are in your `CLASSPATH`. For information about using `gnu.regexp` (<http://www.cacas.org/~wes/java/>) or `gnu.rer` (<http://www.crocodile.org/~sts/Rex/>) with Ant, see this (<http://marc.theaimsgroup.com/?l=ant-devm;=97550753813481w;=2>) article.

Ant will choose the regular-expression library based on the following algorithm:

- If the system property `ant.regexp.matcherimpl` has been set, it is taken as the name of the class implementing `org.apache.tools.ant.util.regexp.RegexpMatcher` that should be used.
- If it has not been set, first try the JDK 1.4 classes, then jakarta-ORO and finally try jakarta-regexp.

Examples:

```
<mapper type="regexp" from="^(.*)\.java$$" to="\1.java.bak"/>
```

Source file name	Target file name
A.java	A.java.bak
foo/bar/B.java	foo/bar/B.java.bak
C.properties	ignored
Classes/dir/dir2/A.properties	ignored

```
<mapper type="regexp" from="^(.*)/([^/]+)/([^/]*).$$" to="\1/\2/\2-\3"/>
```

Source file name	Target file name
A.java	ignored
foo/bar/B.java	foo/bar/bar-B.java
C.properties	ignored
Classes/dir/dir2/A.properties	Classes/dir/dir2/dir2-A.properties

```
<mapper type="regexp" from="^(.*)\.(.*)$$" to="\2.\1"/>
```

Source file name	Target file name
A.java	java.A
foo/bar/B.java	java.foo/bar/B
C.properties	properties.C
Classes/dir/dir2/A.properties	properties.Classes/dir/dir2/A

5.12 Common Attributes of all Tasks

All tasks share the following attributes:

Attribute	Description	Required
id	Unique identifier for this task instance, can be used to reference this task in scripts.	No
taskname	A different name for this task instance - will show up in the logging output.	No
description	Room for your comments	No

6 Core Tasks

6.1 Overview of Ant core tasks

Given the large number of tasks available to a build designer, it may be difficult to get an overall view of what each task can do. The following table provides a short description of each task and a link to the complete documentation.

Task Name	Description
Ant [p 37]	Runs Ant on a supplied buildfile. This can be used to build subprojects.
AntCall [p 38]	Call another target within the same build-file optionally specifying some properties.
AntStructure [p 40]	Generates a DTD for Ant buildfiles which contains information about all tasks currently known to Ant.
Apply/ExecOn [p 41]	Executes a system command. When the <code>os</code> attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.
Available [p 45]	Sets a property if a resource is available at runtime. This resource can be a file, a directory, a class in the classpath, or a JVM system resource.
Chmod [p 47]	Changes the permissions of a file or all files inside specified directories. Right now it has effect only under Unix. The permissions are also UNIX style, like the argument for the <code>chmod</code> command.
Condition [p 49]	Sets a property if a certain condition holds true - this is a generalization of Available [p 45] and Uptodate [p 142] .
Copy [p 52]	Copies a file or Fileset to a new file or directory.
Copydir [p 54]	<i>This task has been deprecated. Use the Copy task instead.</i>
Copyfile [p 56]	<i>This task has been deprecated. Use the Copy task instead.</i>
Cvs [p 57]	Handles packages/modules retrieved from a CVS (http://www.cvshome.org/) repository.
CVSPass [p 59]	Adds entries to a <code>.cvspass</code> file. Adding entries to this file has the same affect as a cvs login command.
Delete [p 60]	Deletes either a single file, all files in a specified directory and its sub-directories, or a set of files specified by one or more FileSet [p 25] s.
Deltree [p 62]	<i>This task has been deprecated. Use the Delete task instead.</i>

Dependset [p 63]	The dependset task compares a set of source files with a set of target files. If any of the source files is more recent than any of the target files, all of the target files are removed.
Ear [p 65]	An extension of the Jar [p 80] task with special treatment for files that should end up in an Enterprise Application archive.
Echo [p 67]	Echoes a message to System.out or a file.
Exec [p 68]	Executes a system command. When the os attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.
Fail [p 70]	Exits the current build (just throwing a BuildException), optionally printing additional information.
Filter [p 71]	Sets a token filter for this project or read multiple token filter from an input file and sets these as filters. Token filters are used by all tasks that perform file copying operations through the Project commodity methods.
FixCRLF [p 72]	Modifies a file to add or remove tabs, carriage returns, linefeeds, and EOF characters.
GenKey [p 76]	Generates a key in keystore.
Get [p 77]	Gets a file from a URL.
GUnzip [p 78]	Expands a GZip file.
GZip [p 79]	GZips a file.
Jar [p 80]	Jars a set of files.
Java [p 83]	Executes a Java class within the running (Ant) VM or forks another VM if specified.
Javac [p 85]	Compiles a source tree within the running (Ant) VM.
Javadoc/Javadoc2 [p 90]	Generates code documentation using the javadoc tool. <i>The Javadoc2 task is deprecated. Use the Javadoc task instead.</i>
Mail [p 97]	A task to send SMTP email.
Mkdir [p 98]	Creates a directory. Also non-existent parent directories are created, when necessary.
Move [p 99]	Moves a file to a new file or directory, or sets of files to a new directory.
Parallel [p 101]	Parallel is a container task - it can contain other Ant tasks. Each nested task within the parallel task will be executed in its own thread.
Patch [p 103]	Applies a diff file to originals.

PathConvert [p 104]	Converts a nested path, path reference, or fileset reference to the form usable on a specified platform and stores the result in a given property.
Property [p 107]	Sets a property (by name and value), or set of properties (from file or resource) in the project.
Record [p 110]	A recorder is a listener to the current build process that records the output to a file.
Rename [p 112]	<i>This task has been deprecated. Use the Move task instead.</i>
Replace [p 113]	Replace is a directory based task for replacing the occurrence of a given string with another string in selected file.
Rmic [p 116]	Runs the rmic compiler for a certain class.
Sequential [p 119]	Sequential is a container task - it can contain other Ant tasks. The nested tasks are simply executed in sequence. Sequential's primary use is to support the sequential execution of a subset of tasks within the parallel [p 101] task
SignJar [p 120]	Signs a jar or zip file with the javasign command line tool.
Sleep [p 121]	A task for sleeping a short period of time, useful when a build or deployment process requires an interval between tasks.
Sql [p 122]	Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the <i>src</i> attribute or from between the enclosing SQL tags.
Style [p 126]	Process a set of documents via XSLT.
Tar [p 129]	Creates a tar archive.
Taskdef [p 132]	Adds a task definition to the current project, such that this new task can be used in the current project.
Touch [p 133]	Changes the modification time of a file and possibly creates it at the same time.
TStamp [p 134]	Sets the DSTAMP, TSTAMP, and TODAY properties in the current project based on the current date and time.
Typedef [p 136]	Adds a data type definition to the current project, such that this new type can be used in the current project.
Unjar [p 141]	Unzips a jarfile.
Untar [p 139]	Untars a tarfile.
Unwar [p 141]	Unzips a warfile.
Unzip [p 141]	Unzips a zipfile.
Uptodate [p 142]	Sets a property if a target files are more up to date than a set of Source files.

War [p 143]	An extension of the Jar [p 80] task with special treatment for files that should end up in the <code>WEB-INF/lib</code> , <code>WEB-INF/classes</code> or <code>WEB-INF</code> directories of the Web Application Archive.
Zip [p 146]	Creates a zipfile.

6.2 Ant

6.2.1 Description

Runs Ant on a supplied buildfile. This can be used to build subprojects.

When the *antfile* attribute is omitted, the file "build.xml" in the supplied directory (*dir* attribute) is used.

If no target attribute is supplied, the default target of the new project is used.

By default, all of the properties of the current project will be available in the new project. Alternatively, you can set the *inheritAll* attribute to `false` and only "user" properties (i.e., those passed on the command-line) will be passed to the new project. In either case, the set of properties passed to the new project will override the properties that are set in the new project (See also the property task [p 107]).

You can also set properties in the new project from the old project by using nested property tags. These properties are always passed regardless of the setting of *inheritAll*. This allows you to parameterize your subprojects.

6.2.2 Parameters

Attribute	Description	Required
antfile	the buildfile to use. Defaults to "build.xml".	No
dir	the directory to use as a basedir for the new Ant project. Defaults to the current directory.	No
target	the target of the new Ant project that should be executed.	No
output	Filename to write the ant output to.	No
inheritAll	If <code>true</code> , pass all properties to the new Ant project. Defaults to <code>true</code> .	No

6.2.3 Examples

```
<ant antfile="subproject/subbuild.xml" dir="subproject" target="compile"/>

<ant dir="subproject"/>

<ant antfile="subproject/property_based_subbuild.xml">
  <property name="param1" value="version 1.x"/>
  <property file="config/subproject/default.properties"/>
</ant>

<ant inheritAll="false" antfile="subproject/subbuild.xml">
  <property name="output.type" value="html"/>
</ant>
```

6.3 AntCall

6.3.1 Description

Call another target within the same build-file optionally specifying some properties (param's in this context)

By default, all of the properties of the current project will be available in the new project. Alternatively, you can set the *inheritAll* attribute to `false` and only "user" properties (i.e., those passed on the command-line) will be passed to the new project. In either case, the set of properties passed to the new project will override the properties that are set in the new project (See also the property task [p 107]).

You can also set properties in the new project from the old project by using nested param tags. These properties are always passed regardless of the setting of *inheritAll*. This allows you to parameterize your subprojects.

When a target is invoked by antcall, all of its dependent targets will also be called within the context of any new parameters. For example, if the target "doSomethingElse" depended on the target "init", then the *antcall* of "doSomethingElse" will call "init" during the call. Of course, any properties defined in the antcall task or inherited from the calling target will be fixed and not overridable in the init task -or indeed in the "doSomethingElse" task.

6.3.2 Parameters

Attribute	Description	Required
target	The target to execute.	Yes
inheritAll	If <code>true</code> , pass all properties to the new Ant project. Defaults to <code>true</code> .	No

6.3.3 Parameters specified as nested elements

6.3.3.1 param

Specifies the properties to set before running the specified target. See property [p 107] for usage guidelines.

6.3.4 Examples

```
<target name="default">
  <antcall target="doSomethingElse">
    <param name="param1" value="value"/>
  </antcall>
</target>

<target name="doSomethingElse">
  <echo message="param1=${param1}"/>
</target>
```

Will run the target 'doSomethingElse' and echo 'param1=value'.

6.4 AntStructure

6.4.1 Description

Generates a DTD for Ant buildfiles which contains information about all tasks currently known to Ant.

Note that the DTD generated by this task is incomplete, you can always add XML entities using `<taskdef>` [p 132] . See here (<http://www.sdv.fr/pages/casa/html/ant-dtd.en.html>) for a way to get around this problem.

This task doesn't know about required attributes, all will be listed as `#IMPLIED`.

6.4.2 Parameters

Attribute	Description	Required
output	file to write the DTD to.	Yes

6.4.3 Examples

```
<antstructure output="project.dtd"/>
```

6.5 Apply/ExecOn

The name `execon` is deprecated and only kept for backwards compatibility.

6.5.1 Description

Executes a system command. When the `os` attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

The files and/or directories of a number of FileSet [p 25] s are passed as arguments to the system command.

If you specify a nested mapper [p 28] and the `destdir` attribute, the timestamp of each source file is compared to the timestamp of a target file which is defined by the nested mapper element and searched for in the given `destdir`.

At least one fileset is required, you must not specify more than one mapper.

6.5.2 Parameters

Attribute	Description	Required
executable	the command to execute without any command line arguments.	Yes
dest	the directory where the <apply> expects the target files will be placed by the command, when it is executed.	Yes, if you specify a nested mapper
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed.	No
output	the file to which the output of the command should be redirected.	No
outputproperty	the name of a property in which the output of the command should be stored.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	No
skipemptyfilesets	Don't run the command, if no source files have been found or are newer than their corresponding target files.	No, default is <i>false</i>
parallel	Run the command only once, appending all files as arguments. If false, command will be executed once for every file. Defaults to false.	No
type	One of <i>file</i> , <i>dir</i> or <i>both</i> . If set to <i>file</i> , only the names of plain files will be sent to the command. If set to <i>dir</i> , only the names of directories are considered.	No, default is <i>file</i>
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is <i>false</i>
vmlauncher	Run command using the Java VM's execution facilities where available. If set to false the underlying OS's shell, either directly or through the antRun scripts, will be used. Under some operating systems, this gives access to facilities not normally available through the VM including, under Windows, being able to execute scripts, rather than their associated interpreter.	No, default is <i>true</i>

6.5.3 Parameters specified as nested elements

6.5.3.1 fileset

You can use any number of nested <fileset> elements to define the files for this task and refer to <fileset>s defined elsewhere.

6.5.3.2 arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments [p 14] .

6.5.3.3 srcfile

By default the file names of the source files will be added to the end of the command line. If you need to place it somewhere different, use a nested `<srcfile>` element between your `<arg>` elements to mark the insertion point.

6.5.3.4 targetfile

`<targetfile>` is similar to `<srcfile>` and marks the position of the target filename on the command line. If omitted, the target filenames will not be added to the command line at all. This element can only be specified, if you also define a nested mapper and the `destdir` attribute.

6.5.3.5 env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements. See the description in the section about `exec` [p 69]

Please note that the environment of the current Ant process is **not** passed to the system command if you specify variables using `<env>`.

6.5.4 Examples

```
<apply executable="ls" >
  <arg value="-l"/>
  <fileset dir="/tmp">
    <patternset>
      <exclude name="**/*.txt"/>
    </patternset>
  </fileset>
  <fileset refid="other.files"/>
</apply>
```

invokes `ls -l`, adding the absolute filenames of all files below `/tmp` not ending in `.txt` and all files of the FileSet with id `other.files` to the command line.

```
<apply executable="somecommand" parallel="false" >
  <arg value="arg1"/>
  <srcfile/>
  <arg value="arg2"/>
  <fileset dir="/tmp"/>
</apply>
```

invokes `somecommand arg1 SOURCEFILENAME arg2` for each file in `/tmp` replacing `SOURCEFILENAME` with the absolute filename of each file in turn. If `parallel` had been set to true, `SOURCEFILENAME` would be replaced with the absolute filenames of all files separated by spaces.

```
<apply executable="cc" dest="src/C" parallel="false">
  <arg value="-c"/>
  <arg value="-o"/>
  <targetfile/>
  <srcfile/>
  <fileset dir="src/C" includes="*.c"/>
  <mapper type="glob" from="*.c" to="*.o"/>
</apply>
```

invokes `CC -C -o TARGETFILE SOURCEFILE` for each `.C` file that is newer than the corresponding `.O`, replacing `TARGETFILE` with the absolute filename of the `.O` and `SOURCEFILE` with the absolute name of the `.C` file.

6.6 Available

6.6.1 Description

Sets a property if a resource is available at runtime. This resource can be a file, a directory, a class in the classpath, or a JVM system resource.

If the resource is present, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the `value` attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on system parameters.

6.6.2 Parameters

Attribute	Description	Required
property	The name of the property to set.	Yes
value	The value to set the property to. Defaults to "true".	No
classname	The class to look for in the classpath.	Yes
file	The file to look for.	
resource	The resource to look for in the JVM.	
classpath	The classpath to use when looking up <code>classname</code> or <code>resource</code> .	No
filepath	The path to use when looking up <code>file</code> .	No
classpathref	The classpath to use, given as a reference [p 15] to a path defined elsewhere.	No
type	The type of <code>file</code> to look for, either a directory (<code>type="dir"</code>) or a file (<code>type="file"</code>). If not set, the property will be set if the name specified in the <code>file</code> attribute exists as either a file or a directory.	No

6.6.3 Parameters specified as nested elements

6.6.3.1 classpath

Available's `classpath` attribute is a path-like structure [p 13] and can also be set via a nested `<classpath>` element.

6.6.3.2 filepath

Available's `filepath` attribute is a path-like structure [p 13] and can also be set via a nested `<filepath>` element.

6.6.4 Examples

```
<available classname="org.whatever.Myclass" property="Myclass.present"/>
```

sets the `Myclass.present` property to the value "true" if the class `org.whatever.Myclass` is found in Ant's classpath.

```
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<available file="{jaxp.jar}" property="jaxp.jar.present"/>
```

sets the `jaxp.jar.present` property to the value "true" if the file `./lib/jaxp11/jaxp.jar` is found.

```
<available file="/usr/local/lib" type="dir" property="local.lib.present"/>
```

sets the `local.lib.present` property to the value "true" if the directory `/usr/local/lib` is found.

```
...in project ...
<property name="jaxp.jar" value="./lib/jaxp11/jaxp.jar"/>
<path id="jaxp" location="{jaxp.jar}"/>
...in target ...
<available classname="javax.xml.transform.Transformer" classpathref="jaxp" property="jaxp11.present"/>
```

sets the `jaxp11.present` property to the value "true" if the class `javax.xml.transform.Transformer` is found in the classpath referenced by `jaxp` (in this case, `./lib/jaxp11/jaxp.jar`).

```
<available property="have.extras" resource="extratasks.properties">
  <classpath>
    <pathelement location="/usr/local/ant/extra.jar"/>
  </classpath>
</available>
```

sets the `have.extras` property to the value "true" if the resource-file `extratasks.properties` is found.

6.7 Chmod

6.7.1 Description

Changes the permissions of a file or all files inside specified directories. Right now it has effect only under Unix. The permissions are also UNIX style, like the argument for the chmod command.

See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task holds an implicit FileSet [p 25] and supports all of FileSet's attributes and nested elements directly. More FileSets can be specified using nested `<fileset>` elements.

6.7.2 Parameters

Attribute	Description	Required
file	the file or single directory of which the permissions must be changed.	exactly one of the two or nested <code><fileset></code> elements.
dir	the directory which holds the files whose permissions must be changed.	
perm	the new permissions.	Yes
includes	comma separated list of patterns of files that must be included.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
parallel	process all specified files using a single <code>chmod</code> command. Defaults to true.	No
type	One of <i>file</i> , <i>dir</i> or <i>both</i> . If set to <i>file</i> , only the permissions of plain files are going to be changed. If set to <i>dir</i> , only the directories are considered.	No, default is <i>file</i>

6.7.3 Examples

```
<chmod file="${dist}/start.sh" perm="ugo+rx"/>
```

makes the "start.sh" file readable and executable for anyone on a UNIX system.

```
<chmod dir="${dist}/bin" perm="ugo+rx" includes="**/*.sh"/>
```

makes all ".sh" files below `${dist}/bin` readable and executable for anyone on a UNIX system.

```
<chmod perm="g+w">
  <fileset dir="shared/sources1">
    <exclude name="**/trial/**"/>
  </fileset>
  <fileset refid="other.shared.sources"/>
</chmod>
```

makes all files below `shared/sources1` (except those below any directory named `trial`) writable for members of the same group on a UNIX system. In addition all files belonging to a FileSet with `id other.shared.sources` get the same permissions.

6.8 Condition

6.8.1 Description

Sets a property if a certain condition holds true - this is a generalization of Available [p 45] and Uptodate [p 142] .

If the condition holds true, the property value is set to true by default; otherwise, the property is not set. You can set the value to something other than the default by specifying the `value` attribute.

Conditions are specified as nested elements [p 49] , you must specify exactly one condition.

6.8.2 Parameters

Attribute	Description	Required
property	The name of the property to set.	Yes
value	The value to set the property to. Defaults to "true".	No

6.8.3 Parameters specified as nested elements

All conditions to test are specified as nested elements.

6.8.3.1 not

The `<not>` element expects exactly one other condition to be nested into this element, negating the result of the condition. It doesn't have any attributes and accepts all nested elements of the condition task as nested elements as well.

6.8.3.2 and

The `<and>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements - all nested elements of the condition task are supported. This condition is true if all of its contained conditions are, conditions will be evaluated in the order they have been specified in the build file.

The `<and>` condition has the same shortcut semantics as the Java `&&` operator, as soon as one of the nested conditions is false, no other condition will be evaluated.

6.8.3.3 or

The `<or>` element doesn't have any attributes and accepts an arbitrary number of conditions as nested elements - all nested elements of the condition task are supported. This condition is true if at least one of its contained conditions is, conditions will be evaluated in the order they have been specified in the build file.

The `<or>` condition has the same shortcut semantics as the Java `||` operator, as soon as one of the nested conditions is true, no other condition will be evaluated.

6.8.3.4 available

This condition is identical to the Available [p 45] task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

6.8.3.5 uptodate

This condition is identical to the Uptodate [p 142] task, all attributes and nested elements of that task are supported, the property and value attributes are redundant and will be ignored.

6.8.3.6 os

Test whether the current operating system is of a given type.

Attribute	Description	Required
family	The name of the operating system family to expect.	No

Supported values for the family attribute are:

- windows (for all versions of Microsoft Windows)
- dos (for all Microsoft DOS based operating systems including Microsoft Windows and OS/2)
- mac (for all Apple Macintosh systems)
- unix (for all Unix and Unix-like operating systems)

6.8.3.7 equals

Tests whether the two given Strings are identical

Attribute	Description	Required
arg1	First string to test.	Yes
arg2	Second string to test.	Yes

6.8.4 Examples

```
<condition property="javamail.complete">
  <and>
    <available classname="javax.activation.DataHandler" />
    <available classname="javax.mail.Transport" />
  </and>
</condition>
```

sets the property `javamail.complete` if both the JavaBeans Activation Framework and JavaMail are available in the classpath.

```
<condition property="isMacOsButNotMacOsX">
  <and>
    <os family="mac" />
    <not>
      <os family="unix" />
    </not>
  </and>
</condition>
```

sets the property `isMacOsButNotMacOsX` if the current operating system is MacOS, but not MacOS X - which Ant considers to be in the Unix family as well.

6.9 Copy

6.9.1 Description

Copies a file or Fileset to a new file or directory. Files are only copied if the source file is newer than the destination file, or when the destination file does not exist. However, you can explicitly overwrite files with the *overwrite* attribute.

FileSet [p 25] s are used to select files to copy. To use a fileset, the *todir* attribute must be set.

6.9.2 Parameters

Attribute	Description	Required
file	The file to copy.	One of either <i>file</i> or at least one nested fileset element.
preservelastmodified	Give the copied files the same last modified time as the original files. Defaults to "no".	No
tofile	The file to copy to.	With the <i>file</i> attribute, either <i>tofile</i> or <i>todir</i> can be used. With nested filesets, only <i>todir</i> is allowed.
todir	The directory to copy to.	
overwrite	Overwrite existing files even if the destination files are newer. Defaults to "no".	No
filtering	Indicates whether token filtering using the global build file filters should take place during the copy. Defaults to "no". Nested filtersets will be used even if this value is "no"	No
flatten	Ignore directory structure of source directory, copy all files into a single directory, specified by the <i>todir</i> attribute. Defaults to "no". Note that you can achieve the same effect by using a flatten mapper [p 29]	No
includeEmptyDirs	Copy empty directories included with the nested FileSet(s). Defaults to "yes".	No

6.9.3 Parameters specified as nested elements

6.9.3.1 fileset

FileSets [p 25] are used to select files to copy. To use a fileset, the *todir* attribute must be set.

6.9.3.2 mapper

You can define file name transformations by using a nested mapper [p 28] element. The default mapper used by `<copy>` is the identity [p 28] .

6.9.3.3 filterset

Filtersets [p 26] are used to replace tokens in files that are copied. To use a filterset just add the nested filterset elements.

6.9.4 Examples

Copy a single file

```
<copy file="myfile.txt" tofile="mycopy.txt"/>
```

Copy a file to a directory

```
<copy file="myfile.txt" todir="../some/dir/tree"/>
```

Copy a directory to another directory

```
<copy todir="../new/dir">
  <fileset dir="src_dir"/>
</copy>
```

Copy a set of files to a directory

```
<copy todir="../dest/dir" >
  <fileset dir="src_dir" >
    <exclude name="**/*.java"/>
  </fileset>
</copy>

<copy todir="../dest/dir" >
  <fileset dir="src_dir" excludes="**/*.java"/>
</copy>
```

Copy a set of files to a directory appending ".bak" to the file name on the fly

```
<copy todir="../backup/dir" >
  <fileset dir="src_dir" />
  <mapper type="glob" from="*" to="*.bak"/>
</copy>
```

Copy a set of files to a replacing @TITLE@ with Foo Bar in all files.

```
<copy todir="../backup/dir" >
  <fileset dir="src_dir" />
  <filterset>
    <filter token="TITLE" value="Foo Bar" />
  </filterset>
</copy>
```

6.10 Copydir

6.10.1 Deprecated

This task has been deprecated. Use the Copy task instead.

6.10.2 Description

Copies a directory tree from the source to the destination.

It is possible to refine the set of files that are being copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `src`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

6.10.3 Parameters

Attribute	Description	Required
src	the directory to copy.	Yes
dest	the directory to copy to.	Yes
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
filtering	indicates whether token filtering should take place during the copy	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the <code>dest</code> attribute (default is <code>false</code>).	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is <code>false</code>).	No

6.10.4 Examples

```
<copydir src="${src}/resources"
        dest="${dist}"
/>
```

copies the directory `${src}/resources` to `${dist}`.

```
<copydir src="${src}/resources"
        dest="${dist}"
        includes="**/*.java"
        excludes="**/Test.java"
/>
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for files with the name `Test.java`.

```
<copydir src="${src}/resources"
        dest="${dist}"
        includes="**/*.java"
        excludes="mypackage/test/**"/>
```

copies the directory `${src}/resources` to `${dist}` recursively. All java files are copied, except for the files under the `mypackage/test` directory.

6.11 Copyfile

6.11.1 *Deprecated*

This task has been deprecated. Use the Copy task instead.

6.11.2 *Description*

Copies a file from the source to the destination. The file is only copied if the source file is newer than the destination file, or when the destination file does not exist.

6.11.3 *Parameters*

Attribute	Description	Required
src	the filename of the file to copy.	Yes
dest	the filename of the file where to copy to.	Yes
filtering	indicates whether token filtering should take place during the copy	No
forceoverwrite	overwrite existing files even if the destination files are newer (default is false).	No

6.11.4 *Examples*

```
<copyfile src="test.java" dest="subdir/test.java" />
```

```
<copyfile src="${src}/index.html" dest="${dist}/help/index.html" />
```

6.12 Cvs

6.12.1 Description

Handles packages/modules retrieved from a CVS (<http://www.cvshome.org/>) repository.

When doing automated builds, the `get` task [p 77] should be preferred over the `checkout` command, because of speed.

6.12.2 Parameters

Attribute	Description	Required
command	the CVS command to execute.	No, default "checkout"
cvsRoot	the CVSROOT variable.	No
dest	the directory where the checked out files should be placed.	No, default is project's basedir.
package	the package/module to check out.	No
tag	the tag of the package/module to check out.	No
date	Use the most recent revision no later than the given date	No
quiet	suppress informational messages.	No, default "false"
noexec	report only, don't change any files.	No, default "false"
output	the file to direct standard output from the command.	No, default output to ANT Log as MSG_INFO.
error	the file to direct standard error from the command.	No, default error to ANT Log as MSG_WARN.
port	Port used by CVS to communicate with the server.	No, default port 2401.
passfile	Password file to read passwords from.	No, default file ~/.cvspass.

6.12.3 Examples

```
<cvs cvsRoot=":pserver:anoncvs@jakarta.apache.org:/home/cvspublic"
  package="jakarta-tools"
  dest="${ws.dir}"
/>
```

checks out the package/module "jakarta-tools" from the CVS repository pointed to by the `cvsRoot` attribute, and stores the files in "\${ws.dir}".

```
<cvs dest="${ws.dir}" command="update"/>
```

updates the package/module that has previously been checked out into "\${ws.dir}".

```
<cvsv command="-q diff -u -N" output="patch.txt"/>
```

silently (-q) creates a file called patch.txt which contains a unified (-u) diff which includes new files added via "cvs add" (-N) and can be used as input to patch.

```
<cvsv command="update -A -d"/>
```

Updates from the head of repository ignoring sticky bits (-A) and creating any new directories as necessary (-d).

Note: the text of the command is passed to cvs "as-is" so any cvs options should appear before the command, and any command options should appear after the command as in the diff example above. See the cvs manual (<http://www.cvshome.org/docs/manual/index.html>) for details, specifically the Guide to CVS commands (http://www.cvshome.org/docs/manual/cvs_16.html)

6.13 cvspass

6.13.1 Description

Adds entries to a .cvspass file. Adding entries to this file has the same affect as a cvs login command.

6.13.2 Parameters

Attribute	Description	Required
cvsroot	the CVS repository to add an entry for.	Yes
password	Password to be added to the password file.	Yes
passfile	Password file to add the entry to.	No, default is ~/.cvspass.

6.13.3 Examples

```
<cvspass cvsroot=":pserver:anoncvs@jakarta.apache.org:/home/cvspublic"  
    password="anoncvs"  
>
```

Adds an entry into the ~/.cvspass password file.

6.14 Delete

6.14.1 Description

Deletes either a single file, all files in a specified directory and its sub-directories, or a set of files specified by one or more FileSet [p 25] s. When specifying a set of files, empty directories are *not* removed by default. To remove empty directories, use the *includeEmptyDirs* attribute.

6.14.2 Parameters

Attribute	Description	Required
file	The file to delete.	at least one of the two
dir	The directory to delete files from.	
verbose	Show name of each deleted file ("true"/"false"). Default is "false" when omitted.	No
quiet	If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error (unless Ant has been invoked with the -verbose or -debug switches). This means that if a file or directory cannot be deleted, then no error is reported. This setting emulates the -f option to the Unix "rm" command. ("true"/"false"). Default is "false" meaning things are "noisy". Setting this to true, implies setting failonerror to false.	No
failonerror	This flag (which is only of relevance if 'quiet' is false), controls whether an error -such as a failure to delete a file- stops the build task, or is merely reported to the screen. The default is "true"	No
includeEmptyDirs	Set to "true" to delete empty directories when using filesets. Default is "false".	No
includes	<i>Deprecated.</i> Comma separated list of patterns of files that must be deleted. All files are in the current directory and any sub-directories are deleted when omitted.	No
includesfile	<i>Deprecated.</i> The name of a file. Each line of this file is taken to be an include pattern	No
excludes	<i>Deprecated.</i> Comma separated list of patterns of files that must be excluded from the deletion list. No files (except default excludes) are excluded when omitted.	No
excludesfile	<i>Deprecated.</i> The name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	<i>Deprecated.</i> Indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

6.14.3 Examples

```
<delete file="/lib/ant.jar"/>
```

deletes the file `/lib/ant.jar`.

```
<delete dir="lib"/>
```

deletes the `lib` directory, including all files and subdirectories of `lib`.

```
<delete>
  <fileset dir="." includes="**/*.bak"/>
</delete>
```

deletes all files with the extension `".bak"`; from the current directory and any sub-directories.

```
<delete includeEmptyDirs="true" >
  <fileset dir="build" />
</delete>
```

deletes all files and subdirectories of `build`, including `build` itself.

6.15 Deltree

6.15.1 *Deprecated*

This task has been deprecated. Use the Delete task instead.

6.15.2 *Description*

Deletes a directory with all its files and subdirectories.

6.15.3 *Parameters*

Attribute	Description	Required
dir	the directory to delete.	Yes

6.15.4 *Examples*

```
<deltree dir="dist"/>
```

deletes the directory `dist`, including its files and subdirectories.

```
<deltree dir="${dist}"/>
```

deletes the directory `${dist}`, including its files and subdirectories.

6.16 DependSet

A task to manage arbitrary dependencies between files.

6.16.1 Description

The dependset task compares a set of source files with a set of target files. If any of the source files is more recent than any of the target files, all of the target files are removed.

Source files and target files are specified via nested FileSets [p 25] and/or nested FileLists [p 24] . Arbitrarily many source and target filesets/filelists may be specified, but at least one filelist/fileset is required for both sources and targets.

Use a FileSet when you want to use wildcard include or exclude patterns and don't care about missing files. Use a FileList when you want to consider the non-existence of a file as if it were out of date. If there are any non-existing files in any source or target FileList, all target files will be removed.

DependSet is useful to capture dependencies that are not or cannot be determined algorithmically. For example, the <style> task only compares the source XML file and XSLT stylesheet against the target file to determine whether to restyle the source. Using dependset you can extend this dependency checking to include a DTD or XSD file as well as other stylesheets imported by the main stylesheet.

6.16.2 Parameters

(none)

6.16.3 Parameters Specified as Nested Elements

6.16.3.1 srcfileset

The nested `srcfileset` element specifies a FileSet [p 25] . All files included in this fileset will be compared against all files included in all of the `targetfileset` filesets and `targetfilelist` filelists. Multiple `srcfileset` filesets may be specified.

6.16.3.2 srcfilelist

The nested `srcfilelist` element specifies a FileList [p 24] . All files included in this filelist will be compared against all files included in all of the `targetfileset` filesets and `targetfilelist` filelists. Multiple `srcfilelist` filelists may be specified.

6.16.3.3 targetfileset

The nested `targetfileset` element specifies a FileSet [p 25] . All files included in this fileset will be compared against all files included in all of the `srcfileset` filesets and `sourcefilelist` filelists, and if any are older, they are all deleted.

6.16.3.4 targetfilelist

The nested `targetfilelist` element specifies a `FileList` [p 24] . All files included in this filelist will be compared against all files included in all of the `srcfileset` filesets and `sourcefilelist` filelists, and if any are older, they are all deleted.

6.16.4 Examples

```
<dependset>
  <srcfilelist
    dir   = "${dtd.dir}"
    files = "paper.dtd,common.dtd"/>
  <srcfilelist
    dir   = "${xsl.dir}"
    files = "common.xsl"/>
  <srcfilelist
    dir   = "${basedir}"
    files = "build.xml"/>
  <targetfileset
    dir      = "${output.dir}"
    includes = "**/*.html"/>
</dependset>
```

In this example derived HTML files in the `${output.dir}` directory will be removed if any are out-of-date with respect to:

1. the DTD of their source XML files
2. a common DTD (imported by the main DTD)
3. a subordinate XSLT stylesheet (imported by the main stylesheet), or
4. the buildfile

If any of the source files in the above example does not exist, all target files will also be removed. To ignore missing source files instead, use filesets instead of filelists for the source files.

6.17 Ear

6.17.1 Description

An extension of the Jar [p 80] task with special treatment for files that should end up in an Enterprise Application archive.

(The Ear task is a shortcut for specifying the particular layout of a EAR file. The same thing can be accomplished by using the *prefix* and *fullpath* attributes of zipfilesets in a Zip or Jar task.)

The extended zipfileset element from the zip task (with attributes *prefix*, *fullpath*, and *src*) is available in the Ear task.

6.17.2 Parameters

Attribute	Description	Required
earfile	the ear-file to create.	Yes
appxml	The deployment descriptor to use (META-INF/application.xml).	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true	No
encoding	The character encoding to use for filenames inside the archive. Defaults to UTF8. It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.	No
filesonly	Store only file entries, defaults to false	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
update	indicates whether to update or overwrite the destination file if it already exists.	No
whenempty	Behavior to use if no files match.	No

6.17.3 *Nested elements*

6.17.3.1 **metainf**

The nested `metainf` element specifies a FileSet [p 25] . All files included in this fileset will end up in the META-INF directory of the ear file. If this fileset includes a file named MANIFEST.MF, the file is ignored and you will get a warning.

6.18 Example

```
<ear earfile="${build.dir}/myapp.ear" appxml="${src.dir}/metadata/application.xml">
  <fileset dir="${build.dir}" includes="*.jar,*.war"/>
</ear>
```

6.19 Echo

6.19.1 Description

Echoes a message to System.out or a file.

6.19.2 Parameters

Attribute	Description	Required
message	the message to echo.	Yes, unless data is included in a character section within this element.
file	the file to write the message to.	No
append	Append to an existing file?	No - default is false.

6.19.3 Examples

```
<echo message="Hello world"/>
```

```
<echo>
This is a longer message stretching over
two lines.
</echo>
```

6.20 Exec

6.20.1 Description

Executes a system command. When the *os* attribute is specified, then the command is only executed when Ant is run on one of the specified operating systems.

6.20.2 Parameters

Attribute	Description	Required
command	the command to execute with all command line arguments. deprecated, use executable and nested <arg> elements instead.	Exactly one of the two.
executable	the command to execute without any command line arguments.	
dir	the directory in which the command should be executed.	No
os	list of Operating Systems on which the command may be executed. If the current OS's name is contained in this list, the command will be executed. The OS's name is determined by the Java Virtual machine and is set in the "os.name" system property.	No
output	the file to which the output of the command should be redirected.	No
outputproperty	the name of a property in which the output of the command should be stored.	No
timeout	Stop the command if it doesn't finish within the specified time (given in milliseconds).	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0. Defaults to false	No
newenvironment	Do not propagate old environment when new environment variables are specified.	No, default is <i>false</i>
vmlauncher	Run command using the Java VM's execution facilities where available. If set to false the underlying OS's shell, either directly or through the antRun scripts, will be used. Under some operating systems, this gives access to facilities not normally available through the VM including, under Windows, being able to execute scripts, rather than their associated interpreter.	No, default is <i>true</i>

6.20.3 Examples

```
<exec dir="${src}" executable="cmd.exe" os="Windows 2000" output="dir.txt">
  <arg line="/c dir"/>
</exec>
```

6.20.4 Parameters specified as nested elements

6.20.4.1 arg

Command line arguments should be specified as nested `<arg>` elements. See Command line arguments [p 14] .

6.20.4.2 env

It is possible to specify environment variables to pass to the system command via nested `<env>` elements.

Please note that the environment of the current Ant process is **not** passed to the system command if you specify variables using `<env>`.

Attribute	Description	Required
key	The name of the environment variable.	Yes
value	The literal value for the environment variable.	Exactly one of these.
path	The value for a PATH like environment variable. You can use ; or : as path separators and Ant will convert it to the platform's local conventions.	
file	The value for the environment variable. Will be replaced by the absolute file-name of the file by Ant.	

6.20.4.2.1 Examples

```
<exec executable="emacs" >
  <env key="DISPLAY" value=":1.0"/>
</exec>
```

starts `emacs` on display 1 of the X Window System.

```
<exec ... >
  <env key="PATH" path="${java.library.path}:${basedir}/bin"/>
</exec>
```

adds `${basedir}/bin` to the `PATH` of the system command.

Note: Although it may work for you to specify arguments using a simple `arg`-element and separate them by spaces it may fail if you switch to a newer version of the JDK. JDK < 1.2 will pass these as separate arguments to the program you are calling, JDK >= 1.2 will pass them as a single argument and cause most calls to fail.

Note2: If you are using Ant on Windows and a new DOS-Window pops up for every command which is executed this may be a problem of the JDK you are using. This problem may occur with all JDK's < 1.2.

6.21 Fail

6.21.1 Description

Exits the current build (just throwing a `BuildException`), optionally printing additional information.

The message of the Exception can be set via the message attribute or character data nested into the element.

6.21.2 Parameters

Attribute	Description	Required
message	A message giving further information on why the build exited	No

6.21.3 Examples

```
<fail/>
```

will exit the current build with no further information given.

```
BUILD FAILED
```

```
build.xml:4: No message
```

```
<fail message="Something wrong here."/>
```

will exit the current build and print something like the following to wherever your output goes:

```
BUILD FAILED
```

```
build.xml:4: Something wrong here.
```

```
<fail>Something wrong here.</fail>
```

will give the same result as above.

6.22 Filter

6.22.1 Description

Sets a token filter for this project or read multiple token filter from an input file and sets these as filters. Token filters are used by all tasks that perform file copying operations through the Project commodity methods.

Note 1: the token string must not contain the separators chars (@).

Note 2: Either token and value attributes must be provided, or only the filtersfile attribute.

6.22.2 Parameters

Attribute	Description	Required
token	the token string without @	Yes*
value	the string that should be put to replace the token when the file is copied	Yes*
filtersfile	The file from which the filters must be read. This file must be a formatted as a property file.	Yes*

* see notes 1 and 2 above parameters table.

6.22.3 Examples

```
<filter token="year" value="2000"/>
<copy todir="${dest.dir}" filtering="true">
  <fileset dir="${src.dir}"/>
</copy>
```

will copy recursively all the files from the *src.dir* directory into the *dest.dir* directory replacing all the occurrences of the string *@year@* with *2000*.

```
<filter filtersfile="deploy_env.properties"/>
```

will read all property entries from the *deploy_env.properties* file and set these as filters.

6.23 FixCRLF

6.23.1 Description

Adjusts a text file to local conventions.

The set of files to be adjusted can be refined with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. Patterns provided through the *includes* or *includesfile* attributes specify files to be included. Patterns provided through the *exclude* or *excludesfile* attribute specify files to be excluded. Additionally, default exclusions can be specified with the *defaultexcludes* attribute. See the section on directory based tasks [p 17] , for details of file inclusion/exclusion patterns and their usage.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

The output file is only written if it is a new file, or if it differs from the existing file. This prevents spurious rebuilds based on unchanged files which have been regenerated by this task.

6.23.2 Parameters

Attribute	Description	Required
srcDir	Where to find the files to be fixed up.	Yes
destDir	Where to place the corrected files. Defaults to srcDir (replacing the original file)	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

eol	Specifies how end-of-line (EOL) characters are to be handled. The EOL characters are CR, LF and the pair CRLF. Valid values for this property are: ● asis: leave EOL characters alone ● cr: convert all EOLs to a single CR ● lf: convert all EOLs to a single LF ● crlf: convert all EOLs to the pair CRLF Default is based on the platform on which you are running this task. For Unix platforms, the default is "lf". For DOS based systems (including Windows), the default is "crlf". For Mac OS, the default is "cr". This is the preferred method for specifying EOL. The "cr" attribute (see below) is now deprecated. <i>N.B.:</i> One special case is recognized. The three characters CR-CR-LF are regarded as a single EOL. Unless this property is specified as "asis", this sequence will be converted into the specified EOL type.	No
cr	Deprecated. Specifies how CR characters are to be handled at end-of-line (EOL). Valid values for this property are: ● asis: leave EOL characters alone. ● add: add a CR before any single LF characters. The intent is to convert all EOLs to the pair CRLF. ● remove: remove all CRs from the file. The intent is to convert all EOLs to a single LF. Default is based on the platform on which you are running this task. For Unix platforms, the default is "remove". For DOS based systems (including Windows), the default is "add". <i>N.B.:</i> One special case is recognized. The three characters CR-CR-LF are regarded as a single EOL. Unless this property is specified as "asis", this sequence will be converted into the specified EOL type.	No
javafiles	Used only in association with the "tab" attribute (see below), this boolean attribute indicates whether the fileset is a set of java source files ("yes"/"no"). Defaults to "no". See notes in section on "tab".	No

tab	Specifies how tab characters are to be handled. Valid values for this property are: ● add: convert sequences of spaces which span a tab stop to tabs ● asis: leave tab and space characters alone ● remove: convert tabs to spaces Default for this parameter is "asis". <i>N.B.:</i> When the attribute "javafiles" (see above) is "true", literal TAB characters occurring within Java string or character constants are never modified. This functionality also requires the recognition of Java-style comments. <i>N.B.:</i> There is an incompatibility between this and the previous version in the handling of white space at the end of lines. This version does not remove trailing whitespace on lines.	No
tablength	TAB character interval. Valid values are between 2 and 80 inclusive. The default for this parameter is 8.	No
eof	Specifies how DOS end of file (control-Z) characters are to be handled. Valid values for this property are: ● add: ensure that there is an EOF character at the end of the file ● asis: leave EOF characters alone ● remove: remove any EOF character found at the end Default is based on the platform on which you are running this task. For Unix platforms, the default is remove. For DOS based systems (including Windows), the default is asis.	No

6.23.3 Examples

```
<fixcrlf srcdir="${src}"
  eol="lf" eof="remove"
  includes="**/*.sh"
/>
```

Replaces EOLs with LF characters and removes eof characters from the shell scripts. Tabs and spaces are left as is.

```
<fixcrlf srcdir="${src}"
  eol="crlf"
  includes="**/*.bat"
/>
```

Replaces all EOLs with cr-lf pairs in the batch files. Tabs and spaces are left as is. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems.

```
<fixcrlf srcdir="${src}"  
  tabs="add"  
  includes="**/Makefile"  
>
```

Sets EOLs according to local OS conventions, and converts sequences of spaces and tabs to the minimal set of spaces and tabs which will maintain spacing within the line. Tabs are set at 8 character intervals. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. Many versions of make require tabs prior to commands.

```
<fixcrlf srcdir="${src}"  
  tabs="remove"  
  tablength="3"  
  eol="lf"  
  javafiles="yes"  
  includes="**/*.java"  
>
```

Converts all EOLs in the included java source files to a single LF. Replace all TAB characters except those in string or character constants with spaces, assuming a tab width of 3. If run on a unix system, any CTRL-Z EOF characters at the end of the file are removed. On DOS/Windows, any such EOF characters will be left untouched.

```
<fixcrlf srcdir="${src}"  
  tabs="remove"  
  includes="**/README*"  
>
```

Sets EOLs according to local OS conventions, and converts all tabs to spaces, assuming a tab width of 8. EOF characters are left alone if run on DOS systems, and are removed if run on Unix systems. You never know what editor a user will use to browse README's.

6.24 GenKey

6.24.1 Description

Generates a key in keystore.

6.24.2 Parameters

Attribute	Description	Required
alias	the alias to add under	Yes.
storepass	password for keystore integrity.	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigalg	the algorithm to use in signing	No
keyalg	the method to use when generating name-value pair	No
verbose	(true false) verbose output when signing	No
dname	The distinguished name for entity	Yes if dname element unspecified
validity	(integer) indicates how many days certificate is valid	No
keysize	(integer) indicates the size of key generated	No

Alternatively you can specify the distinguished name by creating a sub-element named `dname` and populating it with `param` elements that have a name and a value. When using the subelement it is automatically encoded properly and , are replace

The following two examples are identical:

6.24.3 Examples

```
<genkey alias="apache-group" storepass="secret" dname="CN=Ant Group,
OU=Jakarta Division, O=Apache.org, C=US"/>
```

```
<genkey alias="apache-group" storepass="secret" >
  <dname>
    <param name="CN" value="Ant Group"/>
    <param name="OU" value="Jakarta Division"/>
    <param name="O" value="Apache.org"/>
    <param name="C" value="US"/>
  </dname>
</genkey>
```

6.25 Get

6.25.1 Description

Gets a file from a URL. When the verbose option is "on", this task displays a '.' for every 100 Kb retrieved.

This task should be preferred above the CVS task [p 57] when doing automated builds. CVS is significantly slower than loading a compressed archive with http/ftp.

The *usetimestamp* option enables you to control downloads so that the remote file is only fetched if newer than the local copy. If there is no local copy, the download always takes place. When a file is downloaded, the timestamp of the downloaded file is set to the remote timestamp, if the JVM is Java1.2 or later. NB: This timestamp facility only works on downloads using the HTTP protocol.

6.25.2 Parameters

Attribute	Description	Required
src	the URL from which to retrieve a file.	Yes
dest	the file where to store the retrieved file.	Yes
verbose	show verbose progress information ("on"/"off").	No
ignoreerrors	Log errors but don't treat as fatal.	No
usetimestamp	conditionally download a file based on the timestamp of the local copy. HTTP only	No

6.25.3 Examples

```
<get src="http://jakarta.apache.org/" dest="help/index.html"/>
```

Gets the index page of <http://jakarta.apache.org/>, and stores it in the file `help/index.html`.

```
<get src="http://jakarta.apache.org/builds/tomcat/nightly/ant.zip"
    dest="optional.jar"
    verbose="true"
    usetimestamp="true"/>
```

Gets the nightly ant build from the tomcat distribution, if the local copy is missing or out of date. Uses the verbose option for progress information.

6.26 GUnzip

6.26.1 Description

Expands a GZip file.

If *dest* is a directory the name of the destination file is the same as *src* (with the ".gz" extension removed if present). If *dest* is omitted, the parent dir of *src* is taken. The file is only expanded if the source file is newer than the destination file, or when the destination file does not exist.

6.26.2 Parameters

Attribute	Description	Required
src	the file to expand.	Yes
dest	the destination file or directory.	No

6.26.3 Examples

```
<gunzip src="test.tar.gz"/>
```

expands *test.tar.gz* to *test.tar*

```
<gunzip src="test.tar.gz" dest="test2.tar"/>
```

expands *test.tar.gz* to *test2.tar*

```
<gunzip src="test.tar.gz" dest="subdir"/>
```

expands *test.tar.gz* to *subdir/test.tar* (assuming *subdir* is a directory).

6.27 GZip

6.27.1 *Description*

GZips a file.

6.27.2 *Parameters*

Attribute	Description	Required
src	the file to gzip.	Yes
zipfile	the destination file.	Yes

6.27.3 *Examples*

```
<gzip src="test.tar" zipfile="test.tar.gz" />
```

6.28 Jar

6.28.1 Description

Jars a set of files.

The *basedir* attribute is the reference directory from where to jar.

Note that file permissions will not be stored in the resulting jarfile.

It is possible to refine the set of files that are being jarred. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (*dir* becomes *basedir*) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

You can also use nested file sets for more flexibility, and specify multiple ones to merge together different trees of files into one JAR. The extended fileset attributes from the zip task are also available in the jar task. See the Zip [p 146] task for more details and examples.

If the manifest is omitted, a simple one will be supplied by Ant. You should not include META-INF/MANIFEST.MF in your set of files.

The *update* parameter controls what happens if the JAR file already exists. When set to *yes*, the JAR file is updated with the files specified. When set to *no* (the default) the JAR file is overwritten. An example use of this is provided in the Zip task documentation [p 146] .

The *whenempty* parameter controls what happens when no files match. If *create* (the default), the JAR is created anyway with only a manifest. If *skip*, the JAR is not created and a warning is issued. If *fail*, the JAR is not created and the build is halted with an error.

(The Jar task is a shortcut for specifying the manifest file of a JAR file. The same thing can be accomplished by using the *fullpath* attribute of a *zipfileset* in a Zip task. The one difference is that if the *manifest* attribute is not specified, the Jar task will include an empty one for you.)

6.28.2 Parameters

Attribute	Description	Required
jarfile	the jar-file to create.	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true	No
encoding	The character encoding to use for filenames inside the archive. Defaults to UTF8. It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.	No
filesonly	Store only file entries, defaults to false	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
update	indicates whether to update or overwrite the destination file if it already exists.	No
whenempty	Behavior to use if no files match.	No

6.28.3 Nested elements

6.28.3.1 metainf

The nested `metainf` element specifies a FileSet [p 25] . All files included in this fileset will end up in the META-INF directory of the jar file. If this fileset includes a file named `MANIFEST.MF`, the file is ignored and you will get a warning.

6.28.4 Examples

```
<jar jarfile="${dist}/lib/app.jar" basedir="${build}/classes"/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory.

```
<jar jarfile="${dist}/lib/app.jar"
    basedir="${build}/classes"
    excludes="**/Test.class"
/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded.

```
<jar jarfile="${dist}/lib/app.jar"
    basedir="${build}/classes"
    includes="mypackage/test/**"
    excludes="**/Test.class"
/>
```

jars all files in the `${build}/classes` directory into a file called `app.jar` in the `${dist}/lib` directory. Only files under the directory `mypackage/test` are used, and files with the name `Test.class` are excluded.

```
<jar jarfile="${dist}/lib/app.jar">
  <fileset dir="${build}/classes"
    excludes="**/Test.class"
  />
  <fileset dir="${src}/resources"/>
</jar>
```

jars all files in the `${build}/classes` directory and also in the `${src}/resources` directory together into a file called `app.jar` in the `${dist}/lib` directory. Files with the name `Test.class` are excluded. If there are files such as `${build}/classes/mypackage/MyClass.class` and `${src}/resources/mypackage/image.gif`, they will appear in the same directory in the JAR (and thus be considered in the same package by Java).

6.29 Java

6.29.1 Description

Executes a Java class within the running (Ant) VM or forks another VM if specified.

Be careful that the executed class doesn't call `System.exit()`, because it will terminate the VM and thus Ant. In case this happens, it's highly suggested that you set the `fork` attribute so that `System.exit()` stops the other VM and not the one that is currently running Ant.

6.29.2 Parameters

Attribute	Description	Required
classname	the Java class to execute.	Either jar or classname
jar	the location of the jar file to execute (must have a Main-Class entry in the manifest). Fork must be set to true if this option is selected.	Either jar or classname
args	the arguments for the class that is executed. deprecated, use nested <code><arg></code> elements instead.	No
classpath	the classpath to use.	No
classpathref	the classpath to use, given as reference [p 15] to a PATH defined elsewhere.	No
fork	if enabled triggers the class execution in another VM (disabled by default)	No
jvm	the command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by <code>java.lang.Runtime.exec()</code> . Ignored if fork is disabled.	No
jvmargs	the arguments to pass to the forked VM (ignored if fork is disabled). deprecated, use nested <code><jvmarg></code> elements instead.	No
maxmemory	Max amount of memory to allocate to the forked VM (ignored if fork is disabled)	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0. Only available if fork is true.	No
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No
output	Name of a file to write the output to.	No

6.29.3 Parameters specified as nested elements

6.29.3.1 arg and jvmarg

Use nested `<arg>` and `<jvmarg>` elements to specify arguments for the or the forked VM. See Command line arguments [p 14] .

6.29.3.2 sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the class (either ANT's VM or the forked VM). The attributes for this element are the same as for environment variables [p 69] .

6.29.3.3 classpath

Java's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element.

6.29.3.3.1 Example

```
<java classname="test.Main" >
  <arg value="-h"/>
  <classpath>
    <pathelement location="\test.jar"/>
    <pathelement path="${java.class.path}"/>
  </classpath>
</java>
```

6.29.4 Examples

```
<java classname="test.Main"/>

<java classname="test.Main"
  fork="yes" >
  <sysproperty key="DEBUG" value="true"/>
  <arg value="-h"/>
  <jvmarg value="-Xrunhprof:cpu=samples,file=log.txt,depth=3"/>
</java>
```

6.30 Javac

6.30.1 Description

Compiles a Java source tree.

The source and destination directory will be recursively scanned for Java source files to compile. Only Java files that have no corresponding class file or where the class file is older than the java file will be compiled.

Note: Ant uses only the names of the source and class files to find the classes that need a rebuild. It will not scan the source and therefore will have no knowledge about nested classes, classes that are named different from the source file and so on.

The directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled/copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

It is possible to use different compilers. This can be selected with the "build.compiler" property. Here are the choices:-

- classic (the standard compiler of JDK 1.1/1.2) - javac1.1 and javac1.2 can be used as aliases
- modern (the standard compiler of JDK 1.3/1.4) - javac1.3 and javac1.4 can be used as aliases
- jikes (the Jikes (<http://oss.software.ibm.com/developerworks/opensource/jikes/project>) compiler)
- jvc (the Command-Line Compiler from Microsoft's SDK for Java / Visual J++) - microsoft can be used as an alias
- kjc (the kopi (<http://www.dms.at/kopi/>) compiler)
- gcj (the gcj compiler from gcc)
- sj (Symantec java compiler) - symantec can be used as an alias
- extJavac (run either modern or classic in a JVM of its own)

For JDK 1.1/1.2, classic is the default. For JDK 1.3/1.4, modern is the default. If you wish to use a different compiler interface than those supplied, write a class that implements the CompilerAdapter interface (package org.apache.tools.ant.taskdefs.compilers). Supply the full classname in the "build.compiler" property.

The fork attribute overrides the build.compiler setting and expects a JDK1.1 or higher to be set in java.home.

This task will drop all entries that point to non-existent files/directories from the CLASSPATH it passes to the compiler.

6.30.2 Parameters

Attribute	Description	Required
srcdir	location of the java files. (See Notes at the end)	Yes, unless nested <code><src></code> elements are present.
destdir	location to store the class files.	No
includes	comma-separated list of patterns of files that must be included; all files are included when omitted.	No
includesfile	the name of a file that contains include patterns.	No
excludes	comma-separated list of patterns of files that must be excluded; no files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file that contains exclude patterns.	No
defaultexcludes	indicates whether default excludes should be used (yes no); default excludes are used when omitted.	No
classpath	the classpath to use.	No
bootclasspath	location of bootstrap class files.	No
classpathref	the classpath to use, given as a reference [p 15] to a PATH defined elsewhere.	No
bootclasspathref	location of bootstrap class files, given as a reference [p 15] to a PATH defined elsewhere.	No
extdirs	location of installed extensions.	No
encoding	encoding of source files. (gcj doesn't support this option yet)	No
nowarn	indicates whether -nowarn switch should be passed to the compiler; defaults to <code>off</code> .	No
debug	indicates whether source should be compiled with debug information; defaults to <code>off</code> .	No
optimize	indicates whether source should be compiled with optimization; defaults to <code>off</code> .	No
deprecation	indicates whether source should be compiled with deprecation information; defaults to <code>off</code> .	No
target	generate class files for specific VM version (e.g., 1.1 or 1.2).	No
verbose	asks the compiler for verbose output.	No

depend	enables dependency-tracking for compilers that support this (jikes and classic)	No
includeAntRuntime	whether to include the Ant run-time libraries; defaults to yes .	No
includeJavaRuntime	whether to include the default run-time libraries from the executing VM; defaults to no .	No
fork	whether to execute Javac using the JDK compiler externally; defaults to no .	No
memoryInitialSize	the initial size of the memory for the underlying VM, if javac is run externally, ignored otherwise; defaults to the standard VM memory setting. (examples: 83886080, 81920k, or 80m)	No
memoryMaximumSize	the maximum size of the memory for the underlying VM, if javac is run externally, ignored otherwise; defaults to the standard VM memory setting. (examples: 83886080, 81920k, or 80m)	No
failonerror	indicates whether the build will continue even if there are compilation errors; defaults to true .	No

6.30.3 Parameters specified as nested elements

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

6.30.3.1 src, classpath, bootclasspath and extdirs

Javac's `srcdir`, `classpath`, `bootclasspath` and `extdirs` attributes are path-like structures [p 13] and can also be set via nested `<srcgt;`, `<classpathgt;`, `<bootclasspathgt;` and `<extdirsgt;` elements, respectively.

6.30.4 Examples

```
<javac srcdir="${src}"
      destdir="${build}"
      classpath="xyz.jar"
      debug="on"
/>
```

compiles all `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. The classpath used contains `xyz.jar`, and debug information is on.

```
<javac srcdir="${src}"
      destdir="${build}"
      includes="mypackage/p1/**,mypackage/p2/**"
      excludes="mypackage/p1/testpackage/**"
      classpath="xyz.jar"
      debug="on"
/>
```

compiles `.java` files under the `${src}` directory, and stores the `.class` files in the `${build}` directory. The classpath used contains `xyz.jar`, and debug information is on. Only files under `mypackage/p1` and `mypackage/p2` are used. Files in the `mypackage/p1/testpackage` directory are excluded from compilation.

```
<javac srcdir="${src}:${src2}"
      destdir="${build}"
      includes="mypackage/p1/**,mypackage/p2/**"
      excludes="mypackage/p1/testpackage/**"
      classpath="xyz.jar"
      debug="on"
/>
```

is the same as the previous example, with the addition of a second source path, defined by the property `src2`. This can also be represented using nested `<src>` elements as follows:

```
<javac destdir="${build}"
      classpath="xyz.jar"
      debug="on">
  <src path="${src}"/>
  <src path="${src2}"/>
  <include name="mypackage/p1/**"/>
  <include name="mypackage/p2/**"/>
  <exclude name="mypackage/p1/testpackage/**"/>
</javac>
```

Note: If you are using Ant on Windows and a new DOS window pops up for every use of an external compiler, this may be a problem of the JDK you are using. This problem may occur with all JDKs < 1.2.

Note: If you wish to compile only source-files located in some packages below a common root you should not include these packages in the `srcdir`-attribute. Use `include/exclude`-attributes or elements to filter for these packages. If you include part of your package-structure inside the `srcdir`-attribute (or nested `src`-elements) Ant will start to recompile your sources every time you call it.

6.30.5 Jikes Notes

If the environment variable `JIKES_HOME` is set to the location of the jikes compiler, then the standard Ant invocation scripts automatically set `build.compiler` to `"jikes"`. This enables one to use jikes when available, without having to commit the build file to a single choice of compiler.

Jikes also supports some extra options, which can be set by defining properties prior to invoking the task. The ant developers are aware that this is ugly and inflexible -expect a better solution in the future. All the options are boolean, and must be set to `"true"` or `"yes"` to be interpreted as anything other than false; by default `build.compiler.warnings` is `"true"` while all others are `"false"`

build.compiler.emacs	Enable emacs compatible error messages
build.compiler.warnings This property has been deprecated, use the nowarn attribute instead	don't disable warning messages
build.compiler.pedantic	enable pedantic warnings
build.compiler.fulldepend	enable full dependency checking, "+F" in the jikes manual.

6.31 Javadoc/Javadoc2

6.31.1 Description

Generates code documentation using the javadoc tool.

The source directory will be recursively scanned for Java source files to process but only those matching the inclusion rules, and not matching the exclusions rules will be passed to the javadoc tool. This allows wildcards to be used to choose between package names, reducing verbosity and management costs over time. This task, however, has no notion of "changed" files, unlike the javac [p 85] task. This means all packages will be processed each time this task is run. In general, however, this task is used much less frequently.

This task works seamlessly between different javadoc versions (1.1 and 1.2), with the obvious restriction that the 1.2 attributes will be ignored if run in a 1.1 VM.

NOTE: since javadoc calls System.exit(), javadoc cannot be run inside the same VM as ant without breaking functionality. For this reason, this task always forks the VM. This overhead is not significant since javadoc is normally a heavy application and will be called infrequently.

NOTE: the packagelist attribute allows you to specify the list of packages to document outside of the Ant file. It's a much better practice to include everything inside the build.xml file. This option was added in order to make it easier to migrate from regular makefiles, where you would use this option of javadoc. The packages listed in packagelist are not checked, so the task performs even if some packages are missing or broken. Use this option if you wish to convert from an existing makefile. Once things are running you should then switch to the regular notation.

DEPRECATION: the javadoc2 task simply points to the javadoc task and it's there for back compatibility reasons. Since this task will be removed in future versions, you are strongly encouraged to use javadoc [p 90] instead.

6.31.2 Parameters

Attribute	Description	Availability	Required
sourcepath	Specify where to find source files	all	At least one of the two or nested <sourcepath>
sourcepathref	Specify where to find source files by reference [p 15] to a PATH defined elsewhere.	all	
destdir	Destination directory for output files	all	Yes, unless a doclet has been specified.
maxmemory	Max amount of memory to allocate to the javadoc VM	all	No

sourcefiles	Comma separated list of source files	all	at least one of the two or nested <code><source></code> or <code><package></code>
packagenames	Comma separated list of package files (with terminating wildcard)	all	
packageList	The name of a file containing the packages to process	all	No
classpath	Specify where to find user class files	all	No
Bootclasspath	Override location of class files loaded by the bootstrap class loader	1.2	No
classpathref	Specify where to find user class files by reference [p 15] to a PATH defined elsewhere.	all	No
bootclasspathref	Override location of class files loaded by the bootstrap class loader by reference [p 15] to a PATH defined elsewhere.	1.2	No
Extdirs	Override location of installed extensions	1.2	No
Overview	Read overview documentation from HTML file	1.2	No
access	Access mode: one of public , protected , package , or private	all	No (default protected)
Public	Show only public classes and members	all	No
Protected	Show protected/public classes and members (default)	all	No
Package	Show package/protected/public classes and members	all	No
Private	Show all classes and members	all	No
Old	Generate output using JDK 1.1 emulating doclet	1.2	No
Verbose	Output messages about what Javadoc is doing	1.2	No
Locale	Locale to be used, e.g. en_US or en_US_WIN	1.2	No
Encoding	Source file encoding name	all	No
Version	Include <code>@version</code> paragraphs	all	No
Use	Create class and package usage pages	1.2	No
Author	Include <code>@author</code> paragraphs	all	No
Splitindex	Split index into one file per letter	1.2	No
Windowtitle	Browser window title for the documentation (text)	1.2	No
Doctitle	Include title for the package index(first) page (html-code)	1.2	No

Header	Include header text for each page (html-code)	1.2	No
Footer	Include footer text for each page (html-code)	1.2	No
bottom	Include bottom text for each page (html-code)	1.2	No
link	Create links to javadoc output at the given URL	1.2	No
linkoffline	Link to docs at <url> using package list at <url2> - separate the URLs by using a space character.	1.2	No
group	Group specified packages together in overview page. The format is as described below [p 93] .	1.2	No
nodeprecated	Do not include @deprecated information	all	No
nodeprecatedlist	Do not generate deprecated list	1.2	No
notree	Do not generate class hierarchy	all	No
noindex	Do not generate index	all	No
nohelp	Do not generate help link	1.2	No
nonavbar	Do not generate navigation bar	1.2	No
serialwarn	FUTURE: Generate warning about @serial tag	1.2	No
helpfile	FUTURE: Specifies the HTML help file to use	1.2	No
stylesheetfile	Specifies the CSS stylesheet to use	1.2	No
charset	FUTURE: Charset for cross-platform viewing of generated documentation	1.2	No
docencoding	Output file encoding name	1.1	No
doclet	Specifies the class file that starts the doclet used in generating the documentation.	1.2	No
docletpath	Specifies the path to the doclet class file that is specified with the -doclet option.	1.2	No
docletpathref	Specifies the path to the doclet class file that is specified with the -doclet option by reference [p 15] to a PATH defined elsewhere.	1.2	No
additionalparam	Lets you add additional parameters to the javadoc command line. Useful for doclets. Parameters containing spaces need to be quoted using ";.	1.2	No
failonerror	Stop the buildprocess if the command exits with a returncode other than 0.	all	No

excludepackagenames	comma separated list of packages you don't want docs for.	all	No
defaultexcludes	indicates whether default excludes should be used (yes no); default excludes are used when omitted.	all	No
useexternalfile	indicates whether the sourcefile name specified in srcfiles or as nested source elements should be written to a temporary file to make the command line shorter. Also applies to the package names specified via the package-names attribute or nested package elements. (yes no). Default is no.	all	No

6.31.2.1 Format of the group attribute

The arguments are comma-delimited. Each single argument is 2 space-delimited strings, where the first one is the group's title and the second one a colon delimited list of packages.

If you need to specify more than one group, or a group whose title contains a comma or a space character, using nested group elements [p 95] is highly recommended.

E.g.,

```
group="XSLT_Packages org.apache.xalan.xslt*,XPath_Packages org.apache.xalan.xpath*"
```

6.31.3 Parameters specified as nested elements

6.31.3.1 package

Same as one entry in the list given by packagenames.

6.31.3.1.1 Parameters

Attribute	Description	Required
name	The package name (may be a wildcard)	Yes

6.31.3.2 excludepackage

Same as one entry in the list given by excludepackagenames.

6.31.3.2.1 Parameters

Same as for package.

6.31.3.3 source

Same as one entry in the list given by `sourcefiles`.

6.31.3.3.1 Parameters

Attribute	Description	Required
file	The source file to document	Yes

6.31.3.4 doctitle

Same as the `doctitle` attribute, but you can nest text inside the element this way.

6.31.3.5 header

Similar to `<doctitle>`.

6.31.3.6 footer

Similar to `<doctitle>`.

6.31.3.7 bottom

Similar to `<doctitle>`.

6.31.3.8 link

Create link to javadoc output at the given URL. This performs the same role as the `link` and `linkoffline` attributes. You can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

6.31.3.8.1 Parameters

Attribute	Description	Required
href	The URL for the external documentation you wish to link to	Yes
offline	True if this link is not available online at the time of generating the documentation	No
package-list-loc	The location to the directory containing the package-list file for the external documentation	Only if the <code>offline</code> attribute is true

6.31.3.9 group

Separates packages on the overview page into whatever groups you specify, one group per table. This performs the same role as the group attribute. You can use either syntax (or both at once), but with the nested elements you can easily specify multiple occurrences of the arguments.

6.31.3.9.1 Parameters

Attribute	Description	Required
title	Title of the group	Yes, unless nested <code><title></code> given
packages	List of packages to include in that group. Multiple packages are separated with <code>'.'</code> .	Yes, unless nested <code><package></code> s given

The title may be specified as a nested `<title>` element with text contents, and the packages may be listed with nested `<package>` elements as for the main task.

6.31.3.10 doclet

The doclet nested element is used to specify the doclet that javadoc will use to process the input source files. A number of the standard javadoc arguments are actually arguments of the standard doclet. If these are specified in the javadoc task's attributes, they will be passed to the doclet specified in the `<doclet>` nested element. Such attributes should only be specified, therefore, if they can be interpreted by the doclet in use.

If the doclet requires additional parameters, these can be specified with `<param>` elements within the `<doclet>` element. These parameters are restricted to simple strings. An example usage of the doclet element is shown below:

```
<javadoc ...>
  <doclet name="theDoclet"
    path="path/to/theDoclet">
    <param name="-foo" value="foovalue"/>
    <param name="-bar" value="barvalue"/>
  </doclet>
</javadoc>
```

6.31.3.11 sourcepath, classpath and bootclasspath

Javadoc's *sourcepath*, *classpath* and *bootclasspath* attributes are PATH like structure [p 13] and can also be set via nested *sourcepath*, *classpath* and *bootclasspath* elements respectively.

6.31.4 Example

```
<javadoc packagenames="com.dummy.test.*"
        sourcepath="src"
        excludepackagenames="com.dummy.test.doc-files.*"
        defaultexcludes="yes"
        destdir="docs/api"
        author="true"
        version="true"
        use="true"
        windowtitle="Test API">
<doctitle><![CDATA[<h1>Test</h1>]]></doctitle>
<bottom><![CDATA[<i>Copyright &#169; 2000 Dummy Corp. All Rights Reserved.</i>]]></bottom>
<group title="Group 1 Packages" packages="com.dummy.test.a*" />
<group title="Group 2 Packages" packages="com.dummy.test.b*:com.dummy.test.c*" />
<link offline="true" href="http://java.sun.com/products/jdk/1.2/docs/api/" packagelistLoc="C:\tmp" />
<link href="http://developer.java.sun.com/developer/products/xml/docs/api/" />
</javadoc>
```

6.32 Mail

6.32.1 Description

A task to send SMTP email. Text and text files to include in the message body may be specified. To send binary attachments the optional MimeMail [p 226] task should be used instead

6.32.2 Parameters

Attribute	Description	Required
from	Email address of sender.	Yes
tolist	Comma-separated list of recipients.	Yes
message	Message to send in the body of the email.	Yes
files	Filename(s) of text to send in the body of the email. Multiple files are comma-separated.	
mailhost	Host name of the mail server.	No, default to "local-host"
subject	Email subject line.	No

6.32.3 Examples

```
<mail from="me" tolist="you" subject="Results of nightly build"
  files="build.log"/>
```

Sends an eMail from *me* to *you* with a subject of *Results of nightly build* and includes the contents of *build.log* in the body of the message.

6.33 Mkdir

6.33.1 *Description*

Creates a directory. Also non-existent parent directories are created, when necessary.

6.33.2 *Parameters*

Attribute	Description	Required
dir	the directory to create.	Yes

6.33.3 *Examples*

```
<mkdir dir="${dist}"/>
```

creates a directory `${dist}`.

```
<mkdir dir="${dist}/lib"/>
```

creates a directory `${dist}/lib`.

6.34 Move

6.34.1 Description

Moves a file to a new file or directory, or sets of files to a new directory. By default, the destination file is overwritten if it already exists. When *overwrite* is turned off, then files are only moved if the source file is newer than the destination file, or when the destination file does not exist.

FileSet [p 25] s are used to select sets of files to move to the *todir* directory.

6.34.2 Parameters

Attribute	Description	Required
file	the file to move	One of <i>file</i> or at least one nested fileset element
tofile	the file to move to	With the <i>file</i> attribute, either <i>tofile</i> or <i>todir</i> can be used. With a nested fileset, only <i>todir</i> is allowed.
todir	the directory to move to	
overwrite	overwrite existing files even if the destination files are newer (default is "true")	No
filtering	indicates whether token filtering should take place during the move. See the filter [p 71] task for a description of how filters work.	No
flatten	ignore directory structure of source directory, copy all files into a single directory, specified by the <i>todir</i> attribute (default is "false"). Note that you can achieve the same effect by using a flatten mapper [p 29]	No
includeEmptyDirs	Copy empty directories included with the nested FileSet(s). Defaults to "yes".	No

6.34.3 Parameters specified as nested elements

6.34.3.1 mapper

You can define file name transformations by using a nested mapper [p 28] element. The default mapper used by <copy> is the identity [p 28] .

6.34.4 Examples

Move a single file (rename a file)

```
<move file="file.orig" tofile="file.moved"/>
```

Move a single file to a directory

```
<move file="file.orig" todir="dir/to/move/to"/>
```

Move a directory to a new directory

```
<move todir="new/dir/to/move/to">  
  <fileset dir="src/dir"/>  
</move>
```

Move a set of files to a new directory

```
<move todir="some/new/dir" >  
  <fileset dir="my/src/dir" >  
    <include name="**/*.jar"/>  
    <exclude name="**/ant.jar"/>  
  </fileset>  
</move>
```

Append ".bak" to the names of all files in a directory.

```
<move todir="my/src/dir" >  
  <fileset dir="my/src/dir" >  
    <exclude name="**/*.bak"/>  
  </fileset>  
  <mapper type="glob" from="*" to="*.bak"/>  
</move>
```

6.35 Parallel

6.35.1 Description

Parallel is a container task - it can contain other Ant tasks. Each nested task within the parallel task will be executed in its own thread.

Parallel tasks have a number of uses in an Ant build file including:

- Taking advantage of available processing resources to reduce build time
- Testing servers, where the server can be run in one thread and the test harness is run in another thread.

Care must be taken when using multithreading to ensure the tasks within the threads do not interact. For example, two javac compile tasks which write classes into the same destination directory may interact where one tries to read a class for dependency information while the other task is writing the class file. Be sure to avoid these types of interactions within a <parallel> task

The parallel task has no attributes and does not support any nested elements apart from Ant tasks. Any valid Ant task may be embedded within a parallel task, including other parallel tasks.

Note that while the tasks within the parallel task are being run, the main thread will be blocked waiting for all the child threads to complete.

If any of the tasks within the <parallel> task fails, the remaining tasks in other threads will continue to run until all threads have completed. In this situation, the parallel task will also fail.

The parallel task may be combined with the sequential [p 119] task to define sequences of tasks to be executed on each thread within the parallel block

6.35.2 Examples

```
<parallel>
  <wlrn ...>
  <sequential>
    <sleep seconds="30"/>
    <junit ...>
    <wlstop/>
  </sequential>
</parallel>
```

This example represents a typical pattern for testing a server application. In one thread the server is started (the wlrn task). The other thread consists of a three tasks which are performed in sequence. The sleep task is used to give the server time to come up. Another task which is capable of validating that the server is available could be used in place of the sleep task. The test harness is then run. Once the tests are complete, the server is stopped (using wlstop in this example), allowing both threads to complete. The parallel task will also complete at this time and the build will then continue.

```
<parallel>  
  <javac ...> <!-- compiler servlet code -->  
  <wljspc ...> <!-- precompile JSPs -->  
</parallel>
```

This example shows two independent tasks being run to achieve better resource utilization during the build. In this instance, some servlets are being compiled in one thread and a set of JSPs is being precompiled in another. As noted above, you need to be careful that the two tasks are independent, both in terms of their dependencies and in terms of their potential interactions in Ant's external environment.

6.36 Patch

6.36.1 Description

Applies a diff file to originals.

6.36.2 Parameters

Attribute	Description	Required
patchfile	the file that includes the diff output	Yes
originalfile	the file to patch	No, tries to guess it from the diff file
backups	Keep backups of the unpatched files	No
quiet	Work silently unless an error occurs	No
reverse	Assume patch was created with old and new files swapped.	No
ignorewhitespace	Ignore whitespace differences.	No
strip	Strip the smallest prefix containing <i>num</i> leading slashes from filenames.	No

6.36.3 Examples

```
<patch patchfile="module.1.0-1.1.patch"/>
```

applies the diff included in *module.1.0-1.1.patch* to the files in base directory guessing the filename(s) from the diff output.

```
<patch patchfile="module.1.0-1.1.patch" strip="1"/>
```

like above but one leading directory part will be removed. i.e. if the diff output looked like

```
--- a/mod1.0/A  Mon Jun  5 17:28:41 2000
+++ a/mod1.1/A  Mon Jun  5 17:28:49 2000
```

the leading *a/* will be stripped.

6.37 Pathconvert

6.37.1 Description

Converts a nested path, path reference, or fileset reference to the form usable on a specified platform and stores the result in a given property. This operation is useful when script files (batch files or shell scripts) must be generated by the build system and they contain path information that must be properly formatted for the target architecture, not the architecture on which the build is running, or when you need to create a list of files separated by a given character, like a comma or a space.

Prefix maps can be specified to map Windows drive letters to Unix paths and vice versa.

6.37.2 Parameters

Attribute	Description	Required
targetos	The target architecture. Must be one of 'unix' or 'windows'. This is a shorthand mechanism for specifying both <code>pathsep</code> and <code>dirsep</code> according to the specified target architecture.	Must specify one of <code>targetos</code> , <code>pathsep</code> , or <code>dirsep</code> .
dirsep	The character to use as the directory separator in the generated paths.	No, defaults to current JVM <code>File.separator</code>
pathsep	The character to use as the path element separator in the generated paths.	No, defaults to current JVM <code>File.pathSeparator</code>
property	The name of the property in which to place the converted path	Yes
refid	What to convert, given as a reference [p 15] to a PATH or FILESET defined elsewhere	No, if omitted a nested <path> element must be supplied.

6.37.3 Parameters specified as nested elements

6.37.3.1 map

Specifies the mapping of path prefixes between Unix and Windows.

Attribute	Description	Required
from	The prefix to match. Note that this value is case insensitive when the build is running on a windows platform and case sensitive when running on a Unix platform.	Yes
to	The replacement text to use when <i>from</i> is matched.	Yes

Each map element specifies a single replacement map to be applied to the elements of the path being processed. If no map entries are specified, then no path prefix mapping is performed.

*Note that the map elements are applied in the order specified and the only the first matching map element is applied. So, the ordering of your map elements can be important if any *from* values are prefixes of other *from* values.*

6.37.3.2 path

If a path reference is not supplied using the *refid* attribute, then a nested path element must be supplied. See Path-like Structures [p 13] for details.

6.37.4 Examples

In the examples below, assume that the property **wl.home** has the value **d:\weblogic** on Windows and **/weblogic** on Unix.

6.37.4.1 Example 1

```
<path id="wl.path">
  <pathelement location="${wl.home}/lib/weblogicaux.jar" />
  <pathelement location="${wl.home}/classes" />
  <pathelement location="${wl.home}/mssqlserver4/classes" />
  <pathelement location="c:\winnt\System32" />
</path>

<pathconvert targetos="unix" property="newpath" refid="wl.path"/>
```

Assuming **wl.property** has the value **"d:\weblogic"**, will generate the path shown below and store it in the property named **newpath**

```
/weblogic/lib/weblogicaux.jar:/weblogic/classes:/weblogic/mssqlserver4/classes:/WINNT/SYSTEM32
```

Note that the drive letters have been removed. This is the default behavior when no map elements have been specified.

6.37.4.2 Example 2

```
<pathconvert targetos="unix" property="newpath" >
  <path id="wl.path">
    <pathelement location="${wl.home}/lib/weblogicaux.jar" />
    <pathelement location="${wl.home}/classes" />
    <pathelement location="${wl.home}/mssqlserver4/classes" />
    <pathelement location="c:\winnt\System32" />
  </path>
</pathconvert>
```

This generates the exact same path as the previous example. It demonstrates the use of a nested path element.

6.37.4.3 Example 3

```
<pathconvert targetos="unix" property="newpath" refid="wl.path">
  <map from="d:" to="/foo"/>
  <map from="c:" to="/bar"/>
</pathconvert>
```

This example specifies two map entries that will convert path elements that start with `c:` to `/dos` and `d:` to `/`. The resulting path is shown below.

```
/weblogic/lib/weblogicaux.jar:/weblogic/classes:/weblogic/mssqlserver4/classes:/dos/WINNT/SYSTEM32
```

6.37.4.4 Example 4

```
<pathconvert targetos="windows" property="newpath" >
  <path id="wl.path">
    <pathelement location="${wl.home}/lib/weblogicaux.jar" />
    <pathelement location="${wl.home}/classes" />
    <pathelement location="${wl.home}/mssqlserver4/classes" />
    <pathelement location="/dos/winnt/System32" />
  </path>
  <map from="/dos" to="c:\"/>
  <map from="/" to="d:\"/>
</pathconvert>
```

This example, similar to the one above but targetting windows, specifies two map entries that will convert path elements that start with `/dos` to `c:\` and `/` to `d:\`. Note that the order of the map elements was important here since `/` is a prefix of `/dos`. The resulting path is shown below.

```
d:\weblogic\lib\weblogicaux.jar;d:\weblogic\classes;d:\weblogic\mssqlserver4\classes;c:\WINNT\SYSTEM32
```

6.37.4.5 Example 5

```
<fileset dir="${src.dir}" id="src.files">
  <include name="**/*.java"/>
</fileset>

<pathconvert pathsep="," property="javafiles" refid="src.files"/>
```

This example takes the set of files determined by the fileset (all files ending in `.java`), joins them together separated by commas, and places the resulting list into the property `javafiles`. The directory separator is not specified, so it defaults to the appropriate character for the current platform. Such a list could then be used in another task, like `javadoc`, that requires a comma separated list of files.

6.38 Property

6.38.1 Description

Sets a property (by name and value), or set of properties (from file or resource) in the project.

When a property was set by the user, or was a property in a parent project (that started this project with the ant task [p 37]), then this property cannot be set, and will be ignored. This means that properties set outside the current project always override the properties of the current project.

There are five ways to set properties:

- By supplying both the *name* and *value* attribute.
- By supplying both the *name* and *refid* attribute.
- By setting the *file* attribute with the filename of the property file to load. This property file has the format as defined by the file used in the class `java.util.Properties`.
- By setting the *resource* attribute with the resource name of the property file to load. This property file has the format as defined by the file used in the class `java.util.Properties`.
- By setting the *environment* attribute with a prefix to use. Properties will be defined for every environment variable by prefixing the supplied name and a period to the name of the variable.

Although combinations of the three ways are possible, only one should be used at a time. Problems might occur with the order in which properties are set, for instance.

The value part of the properties being set, might contain references to other properties. These references are resolved at the time these properties are set. This also holds for properties loaded from a property file.

A list of predefined properties can be found here [p 11] .

6.38.2 Parameters

Attribute	Description	Required
name	the name of the property to set.	No
value	the value of the property.	One of these, when using the name attribute
location	Sets the property to the absolute filename of the given file. If the value of this attribute is an absolute path, it is left unchanged (with / and \ characters converted to the current platforms conventions). Otherwise it is taken as a path relative to the project's basedir and expanded.	
refid	Reference [p 15] to an object defined elsewhere. Only yields reasonable results for references to PATH like structures [p 13] or properties.	
resource	the resource name of the property file.	One of these, when not using the name attribute
file	the filename of the property file .	
environment	the prefix to use when retrieving environment variables. Thus if you specify environment="myenv" you will be able to access OS-specific environment variables via property names "myenv.PATH" or "myenv.TERM". Note that if you supply a property name with a final "." it will not be doubled. ie environment="myenv." will still allow access of environment variables through "myenv.PATH" and "myenv.TERM". This functionality is currently only implemented on select platforms. Feel free to send patches to increase the number of platforms this functionality is supported on ;)	
classpath	the classpath to use when looking up a resource.	No
classpathref	the classpath to use when looking up a resource, given as reference [p 15] to a PATH defined elsewhere..	No

6.38.3 Parameters specified as nested elements

6.38.3.1 classpath

Property's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element.

6.38.4 Examples

```
<property name="foo.dist" value="dist"/>
```

sets the property `foo.dist` to the value "dist".

```
<property file="foo.properties"/>
```

reads a set of properties from a file called "foo.properties".

```
<property resource="foo.properties"/>
```

reads a set of properties from a resource called "foo.properties".

Note that you can reference a global properties file for all of your Ant builds using the following:

```
<property file="${user.home}/.ant-global.properties"/>
```

since the "user.home" property is defined by the Java virtual machine to be your home directory. This technique is more appropriate for Unix than Windows since the notion of a home directory doesn't exist on Windows. On the JVM that I tested, the home directory on Windows is "C:\". Different JVM implementations may use other values for the home directory on Windows.

```
<property environment="env"/>
<echo message="Number of Processors = ${env.NUMBER_OF_PROCESSORS}"/>
<echo message="ANT_HOME is set to = ${env.ANT_HOME}"/>
```

reads the system environment variables and stores them in properties, prefixed with "env". Note that this only works on *select* operating systems. Two of the values are shown being echoed.

6.39 Record

6.39.1 Description

A recorder is a listener to the current build process that records the output to a file.

Several recorders can exist at the same time. Each recorder is associated with a file. The filename is used as a unique identifier for the recorders. The first call to the recorder task with an unused filename will create a recorder (using the parameters provided) and add it to the listeners of the build. All subsequent calls to the recorder task using this filename will modify that recorder's state (recording or not) or other properties (like logging level).

Some technical issues: the file's print stream is flushed for "finished" events (buildFinished, targetFinished and taskFinished), and is closed on a buildFinished event.

6.39.2 Parameters

Attribute	Description	Required
name	The name of the file this logger is associated with.	yes
action	This tells the logger what to do: should it start recording or stop? The first time that the recorder task is called for this logfile, and if this attribute is not provided, then the default for this attribute is "start". If this attribute is not provided on subsequent calls, then the state remains as previous. [Values = {start stop}, Default = no state change]	no
append	Should the recorder append to a file, or create a new one? This is only applicable the first time this task is called for this file. [Values = {yes no}, Default=yes]	no
loglevel	At what logging level should this recorder instance record to? This is not a once only parameter (like append is) -- you can increase or decrease the logging level as the build process continues. [Values= {error warn info verbose debug}, Default = no change]	no

6.39.3 Examples

The following build.xml snippet is an example of how to use the recorder to record just the `<javac>` task:

```
...
<compile >
  <record name="log.txt" action="start" />
  <javac ...
  <record name="log.txt" action="stop" />
</compile/>
...
```

The following two calls to `<record>` set up two recorders: one to file "records-simple.log" at logging level `info` (the default) and one to file "ISO.log" using logging level of `verbose`.

```
...
<record name="records-simple.log" />
<record name="ISO.log" loglevel="verbose" />
...
```

6.39.4 Notes

There is some functionality that I would like to be able to add in the future. They include things like the following:

Attribute	Description	Required
messageprefix	Whether or not to include the message prefixes (things like the name of the tasks or targets, etc). This has the same effect as the <code>-emacs</code> command line parameter does to the screen output. [yes no]	no
listener	A classname of a build listener to use from this point on instead of the default listener.	no
includetarget	A comma-separated list of targets to automatically record. If this value is "all", then all targets are recorded. [Default = all]	no
excludetarget		no
includetask	A comma-separated list of task to automatically record or not. This could be difficult as it could conflict with the <code>includetarget/excludetarget</code> . (e.g.: <code>includetarget="compile" excludetask="javac"</code> , what should happen?)	no
excludetask		no
action	add greater flexibility to the action attribute. Things like <code>close</code> to close the print stream.	no

6.40 Rename

6.40.1 *Deprecated*

This task has been deprecated. Use the Move task instead.

6.40.2 *Description*

Renames a given file.

6.40.3 *Parameters*

Attribute	Description	Required
src	file to rename.	Yes
dest	new name of the file.	Yes
replace	Enable replacing of existing file (default: on).	No

6.40.4 *Examples*

```
<rename src="foo.jar" dest="${name}-${version}.jar"/>
```

Renames the file `foo.jar` to `${name}-${version}.jar` (assuming `name` and `version` being predefined properties). If a file named `${name}-${version}.jar` already exists, it will be removed prior to renaming `foo.jar`.

6.41 Replace

6.41.1 Description

Replace is a directory based task for replacing the occurrence of a given string with another string in selected file.

If you want to replace a text that crosses line boundaries, you must use a nested `<replacetoken>` element.

6.41.2 Parameters

Attribute	Description	Required
file	file for which the token should be replaced.	Exactly one of the two.
dir	The base directory to use when replacing a token in multiple files.	
token	the token which must be replaced.	Yes, unless a nested <code>replace-token</code> element is used.
value	the new value for the token. When omitted, an empty string ("") is used.	No
summary	Indicates whether a summary of the replace operation should be produced, detailing how many token occurrences and files were processed	No, by default no summary is produced
propertyFile	valid property file from which properties specified using nested <code><replacefilter></code> elements are drawn.	Yes only if <i>property</i> attribute of <code><replacefilter></code> is used.
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

6.41.3 Examples

```
<replace file="${src}/index.html" token="@@@" value="wombat"/>
```

replaces occurrences of the string "@@@" with the string "wombat", in the file `${src}/index.html`.

6.41.4 Parameters specified as nested elements

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

If either the text you want to replace or the replacement text cross line boundaries, you can use nested elements to specify them.

6.41.5 Examples

```
<replace dir="${src}" value="wombat">
  <include name="**/*.html"/>
  <replacetoken><![CDATA[multi line
token]]></replacetoken>
</replace>
```

replaces occurrences of the string "multi line\n token" with the string "wombat", in all HTML files in the directory `${src}`. Where `\n` is the platform specific line separator.

```
<replace file="${src}/index.html">
  <replacetoken><![CDATA[two line
token]]></replacetoken>
  <replacevalue><![CDATA[two line
token]]></replacevalue>
</replace>
```

6.41.5.1 replacefilter

In addition to allowing for multiple replacements, optional nested `<replacefilter>` elements allow replacement values to be extracted from a property file. The name of this file is specified using the `<replace>` attribute *propertyFile*.

Attribute	Description	Required
token	The string to search for.	Yes
value	The replacement string.	Either may be specified, but not both. Both can be omitted, if desired.
property	Name of the property whose value is to serve as the replacement value.	

If neither *value* nor *property* is used, the value provided using the `<replace>` attribute *value* and/or the `<replacevalue>` element is used. If no value was specified using either of these options, the token is replaced with an empty string.

6.41.6 Examples

```
<replace
  file="configure.sh"
  value="defaultvalue"
  propertyFile="source/name.properties" >
  <replacefilter
    token="@token1@" />
  <replacefilter
    token="@token2@"
    value="value2"/>
  <replacefilter
    token="@token3@"
    property="property.key"/>
</replace>
```

In file `configure.sh`, replace all instances of `"@token1@"` with `"defaultvalue"`, all instances of `"@token2@"` with `"value2"`, and all instances of `"@token3@"` with the value of the property `"property.key"`, as it appears in property file `src/name.properties`.

Note: It is possible to use either the `token/<replacetoken>` and `value/<replacevalue>` attributes/elements, the nested `replacefilter` elements, or both in the same operation.

6.42 Rmic

6.42.1 Description

Runs the rmic compiler for a certain class.

Rmic can be run on a single class (as specified with the `classname` attribute) or a number of classes at once (all classes below base that are neither `_Stub` nor `_Skel` classes). If you want to rmic a single class and this class is a class nested into another class, you have to specify the classname in the form `Outer$$Inner` instead of `Outer.Inner`.

It is possible to refine the set of files that are being rmiced. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `base`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

It is possible to use different compilers. This can be selected with the "build.rmic" property. There are three choices:

- sun (the standard compiler of the JDK)
- kaffe (the standard compiler of Kaffe (<http://www.kaffe.org>))
- weblogic

6.42.2 Parameters

Attribute	Description	Required
base	the location to store the compiled files.	Yes
classname	the class for which to run <code>rmic</code> .	No
filtering	indicates whether token filtering should take place	No
sourcebase	Pass the "-keepgenerated" flag to <code>rmic</code> and move the generated source file to the base directory.	No
stubversion	Specify the JDK version for the generated stub code. Specify "1.1" to pass the "-v1.1" option to <code>rmic</code> .	No
classpath	The classpath to use during compilation	No
classpathref	The classpath to use during compilation, given as reference [p 15] to a PATH defined elsewhere	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
verify	check that classes implement Remote before handing them to <code>rmic</code> (default is false)	No
iiop	indicates that portable (RMI/IIOP) stubs should be generated	No
iiopopts	additional arguments for IIOP class generation	No
idl	indicates that IDL output files should be generated	No
idlopts	additional arguments for IDL file generation	No
debug	generate debug info (passes <code>-g</code> to <code>rmic</code>). Defaults to false.	No
includeAntRuntime	whether to include the Ant run-time libraries; defaults to <code>yes</code> .	No
includeJavaRuntime	whether to include the default run-time libraries from the executing VM; defaults to <code>no</code> .	No
extdirs	location of installed extensions.	No

6.42.3 Parameters specified as nested elements

6.42.3.1 classpath and extdirs

`Rmic`'s `classpath` and `extdirs` attributes are PATH like structure [p 13] and can also be set via a nested `classpath` and `extdirs` elements.

6.42.4 Examples

```
<rmic classname="com.xyz.FooBar" base="${build}/classes"/>
```

runs the `rmic` compiler for the class `com.xyz.FooBar`. The compiled files will be stored in the directory `${build}/classes`.

```
<rmic base="${build}/classes" includes="**/Remote*.class"/>
```

runs the `rmic` compiler for all classes with `.class` files below `${build}/classes` whose classname starts with *Remote*. The compiled files will be stored in the directory `${build}/classes`.

6.43 Sequential

6.43.1 Description

Sequential is a container task - it can contain other Ant tasks. The nested tasks are simply executed in sequence. Sequential's primary use is to support the sequential execution of a subset of tasks within the parallel [p 101] task

The sequential task has no attributes and does not support any nested elements apart from Ant tasks. Any valid Ant task may be embedded within the sequential task.

6.43.2 Example

```
<parallel>
  <wlrn ...>
  <sequential>
    <sleep seconds="30"/>
    <junit ...>
    <wlstop/>
  </sequential>
</parallel>
```

This example shows how the sequential task is used to execute three tasks in sequence, while another task is being executed in a separate thread.

6.44 SignJar

6.44.1 Description

Signs a jar or zip file with the `javasign` command line tool.

6.44.2 Parameters

Attribute	Description	Required
jar	the jar file to sign	Yes.
alias	the alias to sign under	Yes.
storepass	password for keystore integrity.	Yes.
keystore	keystore location	No
storetype	keystore type	No
keypass	password for private key (if different)	No
sigfile	name of .SF/.DSA file	No
signedjar	name of signed JAR file	No
verbose	(true false) verbose output when signing	No
internalsf	(true false) include the .SF file inside the signature block	No
sectiononly	(true false) don't compute hash of entire manifest	No

6.44.3 Examples

```
<signjar jar="${dist}/lib/ant.jar" alias="apache-group"
storepass="secret"/>
```

signs the `ant.jar` with alias `"apache-group"` accessing the keystore and private key via `"secret"` password.

6.45 Sleep

6.45.1 Description

A task for sleeping a short period of time, useful when a build or deployment process requires an interval between tasks.

6.45.2 Parameters

Attribute	Description	Required
hours	hours to add to the sleep time	No
minutes	minutes to add to the sleep time	No
seconds	seconds to add to the sleep time	No
milliseconds	milliseconds to add to the sleep time	No
failonerror	flag controlling whether to break the build on an error.	No

The sleep time is the sum of specified values, hours, minutes seconds and milliseconds. A negative value can be supplied to any of them provided the total sleep time is positive

Note that sleep times are always hints to be interpreted by the OS how it feels - small times may either be ignored or rounded up to a minimum timeslice. Note also that the system clocks often have a fairly low granularity too, which complicates measuring how long a sleep actually took.

6.45.3 Examples

```
<sleep milliseconds="10"/>
```

Sleep for about 10 mS.

```
<sleep seconds="2"/>
```

Sleep for about 2 seconds.

```
<sleep hours="1" minutes="-59" seconds="-58"/>
```

Sleep for one hour less 59:58, or two seconds again

```
<sleep />
```

Sleep for no time at all. This may yield the CPU time to another thread or process.

6.46 Sql

6.46.1 Description

Executes a series of SQL statements via JDBC to a database. Statements can either be read in from a text file using the *src* attribute or from between the enclosing SQL tags.

Multiple statements can be provided, separated by semicolons (or the defined *delimiter*). Individual lines within the statements can be commented using either `--`, `//` or `REM` at the start of the line.

The *autocommit* attribute specifies whether auto-commit should be turned on or off whilst executing the statements. If auto-commit is turned on each statement will be executed and committed. If it is turned off the statements will all be executed as one transaction.

The *onerror* attribute specifies how to proceed when an error occurs during the execution of one of the statements. The possible values are: **continue** execution, only show the error; **stop** execution and commit transaction; and **abort** execution and transaction and fail task.

6.46.2 Parameters

Attribute	Description	Required
driver	Class name of the jdbc driver	Yes
url	Database connection url	Yes
userid	Database user name	Yes
password	Database password	Yes
src	File containing SQL statements	Yes, unless statements enclosed within tags
delimiter	String that separates SQL statements	No, default ";"
autocommit	Auto commit flag for database connection (default false)	No, default "false"
print	Print result sets from the statements (default false)	No, default "false"
showheaders	Print headers for result sets from the statements (default true)	No, default "true"
output	Output file for result sets (defaults to System.out)	No (print to System.out by default)
classpath	Classpath used to load driver	No (use system classpath)
onerror	Action to perform when statement fails: continue, stop, abort	No, default "abort"
rdbms	Execute task only if this rdbms	No (no restriction)
version	Execute task only if rdbms version match	No (no restriction)

6.46.3 Parameters specified as nested elements

6.46.3.1 transaction

Use nested `<transaction>` elements to specify multiple blocks of commands to be executed in the same connection but different transactions. This is particularly useful when there are multiple files to execute on the same schema.

Attribute	Description	Required
src	File containing SQL statements	Yes, unless statements enclosed within tags

6.46.3.2 classpath

Sql's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element. It is used to load the JDBC classes.

The

6.46.4 Examples

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
  password="pass"
  src="data.sql"
/>
```

Connects to the database given in *url* as the *sa* user using the *org.database.jdbcDriver* and executes the SQL statements contained within the file *data.sql*

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
  password="pass"
>
insert
into table some_table
values(1,2,3,4);

truncate table some_other_table;
</sql>
```

Connects to the database given in *url* as the *sa* user using the *org.database.jdbcDriver* and executes the two SQL statements inserting data into *some_table* and truncating *some_other_table*

Note that you may want to enclose your statements in `<![CDATA[ ... ]]>` sections so you don't need to escape `<`, `>` & other special characters. For example:

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
  password="pass"
  >![CDATA[

update some_table set column1 = column1 + 1 where column2 < 42;

]]></sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the files data1.sql, data2.sql and data3.sql and then executes the truncate operation on *some_other_table*.

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
  password="pass" >
  <transaction src="data1.sql" />
  <transaction src="data2.sql" />
  <transaction src="data3.sql" />
  <transaction>
    truncate table some_other_table;
  </transaction>
</sql>
```

The following connects to the database given in url as the sa user using the org.database.jdbcDriver and executes the SQL statements contained within the file data.sql, with output piped to outputfile.txt, searching /some/jdbc.jar as well as the system classpath for the driver class.

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
  password="pass"
  src="data.sql"
  print="yes"
  output="outputfile.txt"
  >
<classpath>
  <pathelement location="/some/jdbc.jar">
</classpath>
</sql>
```

The following will only execute if the RDBMS is "oracle" and the version starts with "8.1."

```
<sql
  driver="org.database.jdbcDriver"
  url="jdbc:database-url"
  userid="sa"
```

```
        password="pass"
        src="data.sql"
        rdbms="oracle"
        version="8.1."
    >
    insert
    into table some_table
    values(1,2,3,4);

    truncate table some_other_table;
</sql>
```

6.47 Style

6.47.1 Description

Process a set of documents via XSLT.

This is useful for building views of XML based documentation, or in generating code.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies [p 5] for more information.

It is possible to refine the set of files that are being copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Style supports the use of a `<param>` element which is used to pass values to an `<xsl:param>` declaration.

If you want to use Xalan-J 1 or XSL:P, you also need Ant's optional.jar

6.47.2 Parameters

Attribute	Description	Required
basedir	where to find the source XML file, default is the project's basedir.	No
destdir	directory in which to store the results.	Yes, unless in and out have been specified.
extension	desired file extension to be used for the targets. If not specified, the default is "html".	No
style	name of the stylesheet to use - given either relative to the project's basedir or as an absolute path DEPRECATED - <i>can be specified as a path relative to the basedir attribute of this task as well.</i>	Yes
classpath	the classpath to use when looking up the XSLT processor.	No
classpathref	the classpath to use, given as reference [p 15] to a path defined elsewhere.	No
force	Recreate target files, even if they are newer than their corresponding source files or the stylesheet.	No
processor	name of the XSLT processor to use. Permissible values are "trax" for a TraX compliant processor, "xslp" for the XSL:P processor, "xalan" for the Apache XML Xalan (version 1) processor, or the name of an arbitrary XSLTLiaison class. Defaults to trax, followed by xslp then xalan (in that order). The first one found in your class path is the one that is used. DEPRECATED - <i>XSL:P is deprecated and will be removed in the next version. Use trax or xalan instead..</i>	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
in	specifies a single XML document to be styled. Should be used with the out attribute.	No
out	specifies the output name for the styled result from the in attribute.	No

6.47.3 Parameters specified as nested elements

6.47.3.1 classpath

The classpath to load the processor from can be specified via a nested `<classpath>`, as well - that is, a path [p 13] -like structure.

6.47.3.2 param

Param is used to pass a parameter to the XSL stylesheet.

6.47.3.3 Parameters

Attribute	Description	Required
name	Name of the XSL parameter	Yes
expression	XSL expression to be placed into the param. To pass a text value into the style sheet it needs to be escaped using single quotes.	Yes

6.47.4 Examples

```
<style basedir="doc" destdir="build/doc"
      extension="html" style="style/apache.xml"/>
```

6.47.4.1 Using XSL parameters

```
<style basedir="doc" destdir="build/doc"
      extension="html" style="style/apache.xml">
  <param name="date" expression="'07-01-2000'"/>
</style>
```

This will replace an `xsl:param` definition `<xsl:param name="date"></xsl:param>` with the text value 07-01-2000

6.48 Tar

6.48.1 Description

Creates a tar archive.

The *basedir* attribute is the reference directory from where to tar.

This task is a directory based task [p 17] and, as such, forms an implicit Fileset [p 25] . This defines which files, relative to the *basedir*, will be included in the archive. The tar task supports all the attributes of Fileset to refine the set of files to be included in the implicit fileset.

In addition to the implicit fileset, the tar task supports nested filesets. These filesets are extended to allow control over the access mode, username and groupname to be applied to the tar entries. This is useful, for example, when preparing archives for Unix systems where some files need to have execute permission.

Early versions of tar did not support path lengths greater than 100 characters. Modern versions of tar do so, but in incompatible ways. The behaviour of the tar task when it encounters such paths is controlled by the *longfile* attribute. If the longfile attribute is set to *fail*, any long paths will cause the tar task to fail. If the longfile attribute is set to *truncate*, any long paths will be truncated to the 100 character maximum length prior to adding to the archive. If the value of the longfile attribute is set to *omit* then files containing long paths will be omitted from the archive. Either option ensures that the archive can be untarred by any compliant version of tar. If the loss of path or file information is not acceptable, and it rarely is, longfile may be set to the value *gnu*. The tar task will then produce a GNU tar file which can have arbitrary length paths. Note however, that the resulting archive will only be able to be untarred with GNU tar. The default for the longfile attribute is *warn* which behaves just like the gnu option except that it produces a warning for each file path encountered that does not match the limit.

Note that this task does not perform compression. You might want to use the GZip [p 79] task to prepare a .tar.gz package.

6.48.2 Parameters

Attribute	Description	Required
tarfile	the tar-file to create.	Yes
basedir	the directory from which to zip the files.	No
longfile	Determines how long files (>100 chars) are to be handled. Allowable values are "truncate", "fail", "warn", "omil" and "gnu". Default is "warn".	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

6.48.3 Nested Elements

The tar task supports nested tarfileset [p 25] elements. These are extended Filesets which, in addition to the standard fileset elements, support three additional attributes

Attribute	Description	Required
mode	An 3 digit octal string, specify the user, group and other modes in the standard Unix fashion	No
username	The username for the tar entry. This is not the same as the UID, which is not currently set by the tar task.	No
group	The groupname for the tar entry. This is not the same as the GID, which is not currently set by the tar task.	No

6.48.4 Examples

```
<tar tarfile="${dist}/manual.tar" basedir="htdocs/manual"/>
<gzip zipfile="${dist}/manual.tar.gz" src="${dist}/manual.tar"/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory, then applies the `gzip` task to compress it.

```
<tar tarfile="${dist}/manual.tar"
    basedir="htdocs/manual"
    excludes="mydocs/**, **/todo.html"
/>
```

tars all files in the `htdocs/manual` directory into a file called `manual.tar` in the `${dist}` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<tar longfile="gnu"
  tarfile="${dist.base}/${dist.name}-src.tar" >
  <tarfileset dir="${dist.name}/.." mode="755" username="ant" group="ant">
    <include name="${dist.name}/bootstrap.sh"/>
    <include name="${dist.name}/build.sh"/>
  </tarfileset>
  <tarfileset dir="${dist.name}/.." username="ant" group="ant">
    <include name="${dist.name}/**"/>
    <exclude name="${dist.name}/bootstrap.sh"/>
    <exclude name="${dist.name}/build.sh"/>
  </tarfileset>
</tar>
```

This example shows building a tar which uses the GNU extensions for long paths and where some files need to be marked as executable (mode 755) and the rest are use the default mode (read-write by owner). The first fileset selects just the executable files. The second fileset must exclude the executable files and include all others.

Note: The tar task does not ensure that a file is only selected by one fileset. If the same file is selected by more than one fileset, it will be included in the tar file twice, with the same path.

Note: The patterns in the include and exclude elements are considered to be relative to the corresponding dir attribute as with all other filesets. In the example above, `${dist.name}` is not an absolute path, but a simple name of a directory, so `${dist.name}` is a valid path relative to `${dist.name}/..`.

6.49 Taskdef

6.49.1 Description

Adds a task definition to the current project, such that this new task can be used in the current project. Two attributes are needed, the name that identifies this task uniquely, and the full name of the class (including the packages) that implements this task.

You can also define a group of tasks at once using the file or resource attributes. These attributes point to files in the format of Java property files. Each line defines a single task in the format:

```
taskname=fully.qualified.java.classname
```

Taskdef should be used to add your own tasks to the system. See also "Writing your own task [p 276]".

6.49.2 Parameters

Attribute	Description	Required
name	the name of the task	Yes, unless file or resource have been specified.
classname	the full class name implementing the task	Yes, unless file or resource have been specified.
file	Name of the property file to load taskname/classname pairs from.	No
resource	Name of the property resource to load taskname/classname pairs from.	No
classpath	the classpath to use when looking up classname or resource.	No

6.49.3 Parameters specified as nested elements

6.49.3.1 classpath

Taskdef's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element.

6.49.4 Examples

```
<taskdef name="myjavadoc" classname="com.mydomain.JavadocTask"/>
```

makes a task called myjavadoc available to Ant. The class `com.mydomain.JavadocTask` implements the task.

6.50 Touch

6.50.1 Description

Changes the modification time of a file and possibly creates it at the same time. In addition to working with a single file, this Task can also work a Fileset [p 25] (which also includes directories).

For JDK 1.1 only the creation of new files with a modification time of now works, all other cases will emit a warning.

6.50.2 Parameters

Attribute	Description	Required
file	the name of the file	unless a nested fileset element has been specified.
millis	specifies the new modification time of the file in milliseconds since midnight Jan 1 1970	No
datetime	specifies the new modification time of the file in the format MM/DD/YYYY HH:MM AM_or_PM.	No

If both `millis` and `datetime` are omitted the current time is assumed.

6.50.3 Examples

```
<touch file="myfile"/>
```

creates `myfile` if it doesn't exist and changes the modification time to the current time.

```
<touch file="myfile" datetime="06/28/2000 2:02 pm"/>
```

creates `myfile` if it doesn't exist and changes the modification time to Jun, 28 2000 2:02 pm (14:02 for those used to 24 hour times).

```
<touch datetime="09/10/1974 4:30 pm">
  <fileset dir="src_dir"/>
</touch>
```

changes the modification time to Oct, 09 1974 4:30 pm of all files and directories found in `src_dir`.

6.51 Tstamp

6.51.1 Description

Sets the DSTAMP, TSTAMP, and TODAY properties in the current project. The DSTAMP property is in the format "yyyymmdd", TSTAMP is in the format "hhmm", and TODAY is in the format "month day year".

These properties can be used in the build-file, for instance, to create time-stamped filenames, or used to replace placeholder tags inside documents to indicate, for example, the release date. The best place for this task is probably in an initialization target.

6.51.2 Parameters

Attribute	Description	Required
No parameters		

6.51.3 Nested Elements

The Tstamp task supports a <format> nested element that allows a property to be set to the current date and time in a given format. The date/time patterns are as defined in the Java SimpleDateFormat (<http://java.sun.com/products/jdk/1.2/docs/api/java/text/SimpleDateFormat.html>) class. The format element also allows offsets to be applied to the time to generate different time values.

Attribute	Description	Required
property	The property to receive the date/time string in the given pattern.	Yes
pattern	The date/time pattern to be used. The values are as defined by the Java SimpleDateFormat class.	Yes
offset	The numeric offset to the current time	No
unit	The unit of the offset to be applied to the current time. Valid Values are ● millisecond ● second ● minute ● hour ● day ● week ● month ● year	No
locale	The locale used to create date/time string. The general form is "language, country, variant" but either variant or variant and country may be omitted. For more information please refer to documentation for the Locale (http://java.sun.com/j2se/1.3/docs/api/java/util/Locale.html) class.	No

6.51.4 Examples

```
<tstamp/>
```

sets the standard DSTAMP, TSTAMP, and TODAY properties according to the default formats.

```
<tstamp>
  <format property="TODAY_UK" pattern="d-MMMM-yyyy" locale="en"/>
</tstamp>
```

sets the standard properties as well as the property TODAY_UK with the date/time pattern "d-MMMM-yyyy" using English locale (eg. 21-May-2001).

```
<tstamp>
  <format property="touch.time" pattern="MM/dd/yyyy hh:mm aa"
    offset="-5" unit="hour"/>
</tstamp>
```

Creates a timestamp, in the property touch.time, 5 hours before the current time. The format in this example is suitable for use with the <touch> task

6.52 Typedef

6.52.1 Description

Adds a data type definition to the current project, such that this new type can be used in the current project. Two attributes are needed, the name that identifies this data type uniquely, and the full name of the class (including the packages) that implements this type.

You can also define a group of data types at once using the file or resource attributes. These attributes point to files in the format of Java property files. Each line defines a single data type in the format:

```
typename=fully.qualified.java.classname
```

Typedef should be used to add your own types to the system. Data types are things like paths [p 13] or filesets [p 25] that can be defined at the project level and referenced via their ID attribute.

Custom data types usually need custom tasks to put them to good use.

6.52.2 Parameters

Attribute	Description	Required
name	the name of the data type	Yes, unless file or resource have been specified.
classname	the full class name implementing the data type	Yes, unless file or resource have been specified.
file	Name of the property file to load typename/classname pairs from.	No
resource	Name of the property resource to load typename/classname pairs from.	No
classpath	the classpath to use when looking up classname.	No

6.52.3 Parameters specified as nested elements

6.52.3.1 classpath

Typedef's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element.

6.52.4 Examples

```
<typedef name="urlset" classname="com.mydomain.URLSet"/>
```

makes a data type called `urlset` available to Ant. The class `com.mydomain.URLSet` implements this type.

6.53 Unjar/Unwar/Unzip

6.53.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

6.53.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes
overwrite	Overwrite files, even if they are newer than the corresponding entries in the archive (true or false, default is true).	No

6.53.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

6.54 Untar

6.54.1 Description

Untars a tarfile.

Note: File permissions will not be restored on extracted files.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from tarfile.

The untar task recognizes the long pathname entries used by GNU tar.

6.54.2 Parameters

Attribute	Description	Required
src	tarfile to expand.	Yes
dest	directory where to store the expanded files.	Yes
overwrite	Overwrite files, even if they are newer than the corresponding entries in the archive (true or false, default is true).	No

6.54.3 Examples

```
<gunzip src="tools.tar.gz"/>
<untar src="tools.tar" dest="${tools.home}"/>
```

6.55 Unjar/Unwar/Unzip

6.55.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

6.55.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes
overwrite	Overwrite files, even if they are newer than the corresponding entries in the archive (true or false, default is true).	No

6.55.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

6.56 Unjar/Unwar/Unzip

6.56.1 Description

Unzips a zip-, war- or jarfile.

For JDK 1.1 "last modified time" field is set to current time instead of being carried from zipfile.

File permissions will not be restored on extracted files.

6.56.2 Parameters

Attribute	Description	Required
src	zipfile to expand.	Yes
dest	directory where to store the expanded files.	Yes
overwrite	Overwrite files, even if they are newer than the corresponding entries in the archive (true or false, default is true).	No

6.56.3 Examples

```
<unzip src="${tomcat_src}/tools-src.zip" dest="${tools.home}" />
```

6.57 Uptodate

6.57.1 Description

Sets a property if a target files are more up to date than a set of Source files. Source files are specified by nested `<srcfiles>` elements, these are FileSet [p 25] s, while target files are specified using a nested mapper [p 28] element.

The value part of the property being set is *true* if the timestamp of the target files is more recent than the timestamp of every corresponding source file.

The default behavior is to use a merge mapper [p 29] with the `to` attribute set to the value of the `targetfile` attribute.

Normally, this task is used to set properties that are useful to avoid target execution depending on the relative age of the specified files.

6.57.2 Parameters

Attribute	Description	Required
property	the name of the property to set.	Yes
value	the value to set the property to. Defaults to "true".	No
targetfile	the file for which we want to determine the status.	Yes, unless a nested mapper element is present.

6.57.3 Examples

```
<uptodate property="xmlBuild.notRequired" targetfile="${deploy}\xmlClasses.jar" >
  <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
</uptodate>
```

sets the property *xmlBuild.notRequired* to the value "true" if the *\${deploy}/xmlClasses.jar* is more up to date than any of the DTD files in the *\${src}/xml* directory.

This can be written as

```
<uptodate property="xmlBuild.notRequired" >
  <srcfiles dir= "${src}/xml" includes="**/*.dtd"/>
  <mapper type="merge" to="${deploy}\xmlClasses.jar"/>
</uptodate>
```

as well.

6.58 War

6.58.1 Description

An extension of the Jar [p 80] task with special treatment for files that should end up in the WEB-INF/lib, WEB-INF/classes or WEB-INF directories of the Web Application Archive.

(The War task is a shortcut for specifying the particular layout of a WAR file. The same thing can be accomplished by using the *prefix* and *fullpath* attributes of zipfilesets in a Zip or Jar task.)

The extended zipfileset element from the zip task (with attributes *prefix*, *fullpath*, and *src*) is available in the War task.

6.58.2 Parameters

Attribute	Description	Required
warfile	the war-file to create.	Yes
webxml	The deployment descriptor to use (WEB-INF/web.xml).	Yes
basedir	the directory from which to jar the files.	No
compress	Not only store data but also compress them, defaults to true	No
encoding	The character encoding to use for filenames inside the archive. Defaults to UTF8. It is not recommended to change this value as the created archive will most likely be unreadable for Java otherwise.	No
filesonly	Store only file entries, defaults to false	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
manifest	the manifest file to use.	No
update	indicates whether to update or overwrite the destination file if it already exists.	No
whenempty	Behavior to use if no files match.	No

6.58.3 Nested elements

6.58.3.1 lib

The nested `lib` element specifies a FileSet [p 25] . All files included in this fileset will end up in the `WEB-INF/lib` directory of the war file.

6.58.3.2 classes

The nested `classes` element specifies a FileSet [p 25] . All files included in this fileset will end up in the `WEB-INF/classes` directory of the war file.

6.58.3.3 webinf

The nested `webinf` element specifies a FileSet [p 25] . All files included in this fileset will end up in the `WEB-INF` directory of the war file. If this fileset includes a file named `web.xml`, the file is ignored and you will get a warning.

6.58.3.4 metainf

The nested `metainf` element specifies a FileSet [p 25] . All files included in this fileset will end up in the `META-INF` directory of the war file. If this fileset includes a file named `MANIFEST.MF`, the file is ignored and you will get a warning.

6.58.4 Examples

Assume the following structure in the project's base directory:

```
thirdparty/libs/jdbc1.jar
thirdparty/libs/jdbc2.jar
build/main/com/myco/myapp/Servlet.class
src/metadata/myapp.xml
src/html/myapp/index.html
src/jsp/myapp/front.jsp
src/graphics/images/gifs/small/logo.gif
src/graphics/images/gifs/large/logo.gif
```

then the war file `myapp.war` created with

```
<war warfile="myapp.war" webxml="src/metadata/myapp.xml">
  <fileset dir="src/html/myapp"/>
  <fileset dir="src/jsp/myapp"/>
  <lib dir="thirdparty/libs">
    <exclude name="jdbc1.jar"/>
  </lib>
  <classes dir="build/main"/>
  <zipfileset dir="src/graphics/images/gifs"
    prefix="images"/>
</war>
```

will consist of

```
WEB-INF/web.xml  
WEB-INF/lib/jdbc2.jar  
WEB-INF/classes/com/myco/myapp/Servlet.class  
META-INF/MANIFEST.MF  
index.html  
front.jsp  
images/small/logo.gif  
images/large/logo.gif
```

using Ant's default manifest file. The content of `WEB-INF/web.xml` is identical to `src/meta-data/myapp.xml`.

6.59 Zip

6.59.1 Description

Creates a zipfile.

The *basedir* attribute is the reference directory from where to zip.

Note that file permissions will not be stored in the resulting zipfile.

It is possible to refine the set of files that are being zipped. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `basedir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

Or, you may place within it nested file sets, or references to file sets. In this case `basedir` is optional; the implicit file set is *only used* if `basedir` is set. You may use any mixture of the implicit file set (with `basedir` set, and optional attributes like `includes` and optional subelements like `<include>`); explicit nested `<fileset>` elements so long as at least one fileset total is specified. The ZIP file will only reflect the relative paths of files *within* each fileset. The Zip task and its derivatives know a special form of a fileset named `zipfileset` that has additional attributes (described below).

The `update` parameter controls what happens if the ZIP file already exists. When set to `yes`, the ZIP file is updated with the files specified. (New files are added; old files are replaced with the new versions.) When set to `no` (the default) the ZIP file is overwritten.

The `whenempty` parameter controls what happens when no files match. If `skip` (the default), the ZIP is not created and a warning is issued. If `fail`, the ZIP is not created and the build is halted with an error. If `create`, an empty ZIP file (explicitly zero entries) is created, which should be recognized as such by compliant ZIP manipulation tools.

This task will now use the platform's default character encoding for filenames - this is consistent with the command line ZIP tools, but causes problems if you try to open them from within Java and your filenames contain non US-ASCII characters. Use the `encoding` attribute and set it to UTF8 to create zip files that can safely be read by Java.

6.59.2 Parameters

Attribute	Description	Required
zipfile	the zip-file to create.	Yes
basedir	the directory from which to zip the files.	No
compress	Not only store data but also compress them, defaults to true	No
encoding	The character encoding to use for filenames inside the zip file. For a list of possible values see http://java.sun.com/products/jdk/1.2/docs/guide/international/encoding.doc.html . Defaults to the platform's default character encoding.	No
filesonly	Store only file entries, defaults to false	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
update	indicates whether to update or overwrite the destination file if it already exists.	No
whenempty	Behavior when no files match.	No

6.59.3 Parameters specified as nested elements

6.59.3.1 fileset

The zip task supports any number of nested `<fileset>` [p 25] elements to specify the files to be included in the archive.

6.59.3.2 zipfileset

A `<zipfileset>` has three additional attributes: *prefix*, *fullpath*, and *src*. The *prefix* and *fullpath* attributes modify the location of the files when they are placed inside the archive. If the *prefix* attribute is set, all files in the fileset are prefixed with that path in the archive. If the *fullpath* attribute is set, the file described by the fileset is placed at that exact location in the archive. (The *fullpath* attribute can only be set for filesets that represent a single file. The *prefix* and *fullpath* attributes cannot both be set on the same fileset.) The *src* attribute may be used in place of the *dir* attribute to specify a zip file whose contents will be extracted and included in the archive. As with directories, include and exclude patterns may be used to specify a subset of the zip file for inclusion in the archive.

6.59.4 Examples

```
<zip zipfile="${dist}/manual.zip"
    basedir="htdocs/manual"
/>
```

zips all files in the `htdocs/manual` directory into a file called `manual.zip` in the `${dist}` directory.

```
<zip zipfile="${dist}/manual.zip"
    basedir="htdocs/manual"
    update="true"
/>
```

zips all files in the `htdocs/manual` directory into a file called `manual.zip` in the `${dist}` directory. If `manual.zip` doesn't exist, it is created; otherwise it is updated with the new/changed files.

```
<zip zipfile="${dist}/manual.zip"
    basedir="htdocs/manual"
    excludes="mydocs/**, **/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Files in the directory `mydocs`, or files with the name `todo.html` are excluded.

```
<zip zipfile="${dist}/manual.zip"
    basedir="htdocs/manual"
    includes="api/**/*.html"
    excludes="**/todo.html"
/>
```

zips all files in the `htdocs/manual` directory. Only html files under the directory `api` are zipped, and files with the name `todo.html` are excluded.

```
<zip zipfile="${dist}/manual.zip">
  <fileset dir="htdocs/manual"/>
  <fileset dir="." includes="ChangeLog.txt"/>
</zip>
```

zips all files in the `htdocs/manual` directory, and also adds the file `ChangeLog.txt` in the current directory. `ChangeLog.txt` will be added to the top of the ZIP file, just as if it had been located at `htdocs/manual/ChangeLog.txt`.

```
<zip zipfile="${dist}/manual.zip">
  <zipfileset dir="htdocs/manual" prefix="docs/user-guide"/>
  <zipfileset dir="." includes="ChangeLog27.txt" fullpath="docs/ChangeLog.txt"/>
  <zipfileset src="examples.zip" includes="**/*.html" prefix="docs/examples"/>
</zip>
```

zips all files in the `htdocs/manual` directory into the `docs/user-guide` directory in the archive, adds the file `ChangeLog27.txt` in the current directory as `docs/ChangeLog.txt`, and includes all the html files in `examples.zip` under `docs/examples`. The archive might end up containing the files:

```
docs/user-guide/html/index.html  
docs/ChangeLog.txt  
docs/examples/index.html
```

7 Optional Tasks

7.1 .NET classes for Ant

7.1.1 Vision

Make ant the definitive build tool for .NET projects, just as it is becoming for java projects

7.1.2 Detail

The emergent policy in big java projects is 'IDE for code and debug'; Ant for builds, test and deployment. While IDEs are great for an individual development task, Ant not only rebuilds everything insanely fast, it has great support for modern deployment tasks -ftp upload, EJB servers, SQL commands, all can be done from a relatively simple XML declaration of what you want done. Also in a large project (especially open source) there is no means of dictating which IDE to use. Ant gives IDE independence, cross platform portability and ease of extensibility.

Projects using .net should be able to use the same tool. This will avoid everyone being dependent on Visual Studio.net (if they have it) supplemented with makefiles and batch files. Anyone doing mixed java/dotnet development will be particularly grateful, as they can have a unified build process.

If at sometime in the future bits of .net do appear on different platforms then ant based build files will migrate -once the ant tasks have been migrated and any hard codings of DOS-descendant file system conventions removed from the build.xml files.

7.1.3 Key .net Tasks

1. csc - csharp compiler - *.csc -> output
2. ilasm - IL assembler - *.il ->.exe or .dll
3. ResX - ResXToResources.exe .resx -> .resources
4. al - assembly linker (.dll) ->
5. Signcode - .exe + keys -> .exe
6. Resgen - .properties -> .resources
7. Type Library Exporter (TlbExp.exe) .dll -> .tlb
8. TlbImp.exe Type Library Importer
9. RegAsm.exe Assembly Registration Tool .dll -> .reg | registration

7.1.4 Ant Wrapper for net tasks

Rarely used .net command line tools can be invoked manually. The key tasks to address are the common steps in a build process and those which benefit from the file pattern matching function.

The esoteric tasks can then supported on demand by those who need them. The initial .net tasks should provide a foundation for that demand creation to be simple.

7.1.5 Plan

1. code up C sharp task and distribute for feedback & identify possible aid
2. Use for a real end to end development task
3. Add the next task I need (whatever that is)
4. Refactor to produce a base class for .net tasks
5. Leave it other people write all the other tasks

7.1.6 Risks

- The .net SDK is unstable; the next builds will change the commands, arguments and behaviours. Probability: 100%. Impact: 2 days.
 - Portable tasks turn out to be harder than envisaged (=reworking if/when non Win32 .net build tools appear)
-

7.1.7 Using Ant in .net projects

To use the net tasks one needs

- A Windows box (or any other platform which supports the .net build tools)
 - JavaVM - Java1.1 or later; Java1.3 recommended.
 - Ant downloaded and the environment and path variables set up so that typing 'ant' invokes ant via a batch file.
 - An XML parser for ant. The usual distributions include an appropriate version crimson.jar and jaxp.jar.
 - the .net tasks archive (dotnettasks.jar) included in the ant\lib directory
 - The .net sdk installed with the binaries on the path. (Test: type CSC)
 - A build.xml file to describe the build process
-

7.1.8 Change Log

7.1.8.0.1 Version 0.4

This is the beta-2 revision of the tasks.

1. ILASM: pulled the owner attribute, added keyfile for giving binaries a strong name (MD5 hash of the checksum)
2. CSC: added win32res , noConfig, utf8output, fullpaths
3. CSC:

7.1.8.0.2 Version 0.3

The changes here reflect Beta-1 of the dotnet SDK and experience of use in more complex projects. This build does not work with the older SDK, primarily because the automatic reference feature references libraries only found in the new SDK version.

External changes

- Recursive inclusion of .cs and .il files
- Documentation enhanced, includes examples and details of all parameters
- The csc task automatically includes the common dotnet assemblies, so there is no need to remember to refer to 'System.dll', 'System.Web.Services', etc. This feature can be disabled by setting the 'includeDefaultReferences' flag to false.
- References can also be referred to using the ReferenceFiles parameter, which is an ant path specification. The old 'references' string is still retained.
- An 'extraoptions' attribute enables the build file to include any CSC options which are not explicitly supported in the CSC task.

Internal changes

- Some minor refactoring (move common code a method)
- Application of Jedit's JavaStyle task resulted in a major reshaping of the codebase and the insertion of a blank line every second line. Significant effort was required to revert some (but not all) changes.
- Removed throws clause from methods which can't throw exceptions

The test harness has been expanded to include unicode source file (the build works but the rest of the system has 'issues' with high unicode package and method names)

7.1.8.0.3 Version 0.2

First public edition, added to the ant cvs tree. Tested on the PDC build of the dotnet SDK only, and still immature. The command execution code was refactored out into a 'NetCommand' class for re-use. The Ilasm task was added at this time.

7.1.8.0.4 Version 0.1

Initial proof of concept; very rudimentary support for CSC only.

7.1.9 Task: CSC

This task compiles CSharp source into executables or modules. This task compiles CSharp source into executables or modules. The task will only work on win2K until other platforms support csc.exe or an equivalent. CSC.exe must be on the execute path too.

All parameters are optional: <csc/> should suffice to produce a debug build of all *.cs files. References to external files do require explicit enumeration, so are one of the first attributes to consider adding.

The task is a directory based task, so attributes like **includes**="**/*.cs" and **excludes**="broken.cs" can be used to control the files pulled in. By default, all *.cs files from the project folder down are included in the command. When this happens the output file -if not specified- is taken as the first file in the list, which may be somewhat hard to control. Specifying the output file with '**outfile**' seems prudent.

Attribute	Description	Example Values
additionalModules	Semicolon separated list of modules to refer to	
defaultexcludes	indicates whether default excludes should be used or not	"true"(default) or "false"
definitions	defined constants	"RELEASE;BETA1"
debug	include debug information	"true"(default) or "false"
docFile	name of file for documentation	"doc.xml"
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	
extraOptions	Any extra options which aren't explicitly supported by the CSharp task	"/warnaserror+ /baseaddress:0x12840000"
failOnError	Should a failed compile halt the build?	"true"(default) or "false"
fullpaths	print the full path of files on on errors	
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	
includeDefaultReferences	Flag which when true automatically includes the common assemblies in dotnet beta1, and tells the compiler to link in mscore.dll	"true"(default) or "false"
includesfile	the name of a file. Each line of this file is taken to be an include pattern	
incremental	Incremental build flag. Avoid till it works	"true" or "false"(default)
mainClass	name of main class for executables	"com.example.project.entrypoint"
noConfig	a flag which tells the compiler not to read in the compiler settings files 'csc.rsp' in its bin directory and then the local directory	"true" or "false"(default)
optimize	optimisation flag	"true" or "false"(default)
outputFile	filename of output	"example.exe"

references	Semicolon separated list of dlls to refer to	"mylib.dll;nunit.dll"
referenceFiles	Ant Path description of references to include. Wildcards should work.	
srcDir	source directory (default = project directory)	."
targetType	Type of target	"exe", "module", "winexe" or "library"
unsafe	enable the unsafe keyword	"true" or "false"(default)
utf8output	require all compiler output to be in utf-8 format	"true" or "false"(default)
warnLevel	level of warning currently between 1 and 4 with 4 being the strictest.	"1"-"4"
win32Icon	filename of icon to include	"res/myicon.ico"
win32res	filename of a win32 resource (.RES)file to include This is not a .NET resource, but it what windows is used to.	"res/myapp.res"

Example

```
<csc
    optimize="true"
    debug="false"
    docFile="documentation.xml"
    warnLevel="4"
    unsafe="false"
    targetType="exe"
    incremental="false"
    definitions="RELEASE"
    excludes="src/unicode_class.cs"
    mainClass = "MainApp"
    outputFile="NetApp.exe"
/>
```

7.1.10 Task: ilasm

Task to assemble .net 'Intermediate Language' files. The task will only work on windows until other platforms support csc.exe or an equivalent. ilasm.exe must be on the execute path too.

All parameters are optional: <il/> should suffice to produce a debug build of all *.il files. The option set is roughly compatible with the CSharp class; even though the command line options are only vaguely equivalent. [The low level commands take things like /OUT=file, csc wants /out:file ... /verbose is used some places; /quiet here in ildasm... etc.] It would be nice if someone made all the command line tools consistent (and not as brittle as the java cmdline tools)

The task is a directory based task, so attributes like **includes**="*.il" and **excludes**="broken.il" can be used to control the files pulled in. Each file is built on its own, producing an appropriately named output file unless manually specified with **outfile**

Attribute	Description	Example
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	
debug	include debug information	true (default)
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	
extraOptions	Any extra options which aren't explicitly supported by the ilasm task, primarily because they aren't really documented: use ilasm /? to see them	
failOnError	Should a failed compile halt the build?	"true"(default)
fullpaths	Should error text provide the full path to files	"true"(default)
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	
includesfile	the name of a file. Each line of this file is taken to be an include pattern	
keyfile	the name of a file containing a private key, with which the assembly output is checksummed and then MD5 signed to have a <i>strong name</i>	
listing	Produce a listing (off by default). Listings go to the current output stream	"on", "off" (default)
outputFile	filename of output	"example.exe"
resourceFile	name of resource file to include	"resources.res"
srcDir	source directory (default = project directory)	
targetType	Type of target. library means DLL is output.	"exe"(default), "library"
verbose	output progress messages	"on", "off" (default)

Example

```
<ilasm
  outputFile="app.exe"
  verbose="on"
  listing="on"
  owner="secret"
/>
```

7.2 ANTLR

7.2.1 Description

Invokes the ANTLR (<http://www.antlr.org/>) Translator generator on a grammar file.

To use the ANTLR task, set the *target* attribute to the name of the grammar file to process. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise ANTLR writes the generated files to the directory containing the grammar file.

This task only invokes ANTLR if the grammar file is newer than the generated files.

7.2.2 Parameters

Attribute	Description	Required
target	The grammar file to process.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
fork	Run ANTLR in a separate VM.	No, default is "off"
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No

7.2.3 Example

```
<antlr
  target="etc/java.g"
  outputdirectory="build/src"
/>
```

This invokes ANTLR on grammar file etc/java.g, writing the generated files to build/src.

7.3 Cab

7.3.1 Description

The cab task creates Microsoft cab archive files. It is invoked similar to the jar [p 80] or zip [p 146] tasks. This task will work on Windows using the external cabarc tool (provided by Microsoft) which must be located in your executable path.

To use this task on other platforms you need to download and compile libcabinet from http://trill.cis.fordham.edu/~barbacha/cabinet_library/.

See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of <fileset> (dir becomes basedir) as well as the nested <include>, <exclude> and <patternset> elements.

7.3.2 Parameters

Attribute	Description	Required
cabfile	the name of the cab file to create.	Yes
basedir	the directory to start archiving files from.	Yes
verbose	set to "yes" if you want to see the output from the cabarc tool. defaults to "no".	No
compress	set to "no" to store files without compressing. defaults to "yes".	No
options	use to set additional command-line options for the cabarc tool. should not normally be necessary.	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

7.3.3 Examples

```
<cab cabfile="${dist}/manual.cab"
    basedir="htdocs/manual"
/>
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the \${dist} directory.

```
<cab cabfile="${dist}/manual.cab"
    basedir="htdocs/manual"
    excludes="mydocs/**, **/todo.html"
/>
```

cabs all files in the htdocs/manual directory into a file called manual.cab in the \${dist} directory. Files in the directory mydocs, or files with the name todo.html are excluded.

```
<cab cabfile="${dist}/manual.cab"
    basedir="htdocs/manual"
    includes="api/**/*.html"
    excludes="**/todo.html"
    verbose="yes"
/>
```

Cab all files in the htdocs/manual directory into a file called manual.cab in the \${dist} directory. Only html files under the directory api are archived, and files with the name todo.html are excluded. Output from the cabarc tool is displayed in the build output.

7.4 ClearCase Support

- CCCheckin [p 159]
 - CCCheckout [p 159]
 - CCUnCheckout [p 160]
 - CCUpdate [p 161]
-

7.4.1 CCCheckin

7.4.1.1 Description

Task to perform a Checkin command to ClearCase.

7.4.1.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No
nowarn	Suppress warning messages	No
preservetime	Preserve the modification time	No
keepcopy	Keeps a copy of the file with a .keep extension	No
identical	Allows the file to be checked in even if it is identical to the original	No

7.4.1.3 Examples

```
<cccheckin viewpath="c:/views/viewdir/afile"  
  commentfile="acomment.txt"/>  
  nowarn="true"  
  identical="true"
```

Does a ClearCase *checkin* on the file *c:/views/viewdir/afile*. Comment text from the file *acomment.txt* is added to ClearCase as a comment. All warning messages are suppressed. The file is checked in even if it is *identical* to the original.

7.4.2 CCCheckout

7.4.2.1 Description

Task to perform a Checkout command to ClearCase.

7.4.2.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
reserved	Specifies whether to check out the file as reserved or not	Yes
out	Creates a writable file under a different filename	No
nodata	Checks out the file but does not create an editable file containing its data	No
branch	Specify a branch to check out the file to	No
version	Allows checkout of a version other than main latest	No
nowarn	Suppress warning messages	No
comment	Specify a comment. Only one of comment or commentfile may be used.	No
commentfile	Specify a file containing a comment. Only one of comment or commentfile may be used.	No

7.4.2.3 Examples

```
<cccheckout viewpath="c:/views/viewdir/afile"
  reserved="true"
  branch="abranh"
  nowarn="true"
  comment="Some comment text"/>
```

Does a ClearCase *checkout* on the file *c:/views/viewdir/afile*. It is checked out as *reserved* on branch called *abranh*. All warning messages are suppressed. A *Some comment text* is added to ClearCase as a comment.

7.4.3 CCUnCheckout

7.4.3.1 Description

Task to perform a UnCheckout command to ClearCase.

7.4.3.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
keepcopy	Specifies whether to keep a copy of the file with a .keep extension or not	No

7.4.3.3 Examples

```
<ccuncheckout viewpath="c:/views/viewdir/afile"
  keepcopy="true"/>
```

Does a ClearCase *uncheckout* on the file *c:/views/viewdir/afile*. A copy of the file called *c:/views/viewdir/afile.keep* is kept.

7.4.4 CCUpdate

7.4.4.1 Description

Task to perform an Update command to ClearCase.

7.4.4.2 Parameters

Attribute	Values	Required
viewpath	Path to the ClearCase view file or directory that the command will operate on	No
graphical	Displays a graphical dialog during the update	No
log	Specifies a log file for ClearCase to write to	No
overwrite	Specifies whether to overwrite hijacked files or not	No
rename	Specifies that hijacked files should be renamed with a .keep extension	No
currenttime	Specifies that modification time should be written as the current time. Either currenttime or preservetime can be specified.	No
preservetime	Specifies that modification time should be preserved from the VOB time. Either currenttime or preservetime can be specified.	No

7.4.4.3 Examples

```
<ccupdate viewpath="c:/views/viewdir"
  graphical="false"
  log="log.log"
  overwrite="true"
  currenttime="true"
  rename="false"/>
```

Does a ClearCase *update* on the directory *c:/views/viewdir*. A graphical dialog will be displayed. The output will be logged to *log.log* and it will overwrite any hijacked files. The modified time will be set to the current time.

7.5 Continuous Support

- CCMCheckin [p 162]
- CCMCheckout [p 162]
- CCMCheckinTask [p 163]
- CCMReconfigure [p 163]
- CCMCreateTask [p 164]

These ant tasks are wrappers around Continuous Source Manager. They have been tested with version 5.1 on Windows 2000, but should work on other platforms with ccm installed.

7.5.1 CCMCheckin

7.5.1.1 Description

Task to checkin a file

7.5.1.2 Parameters

Attribute	Values	Required
file	Path to the file that the command will operate on	Yes
comment	Specify a comment. Default is "Checkin" plus the date	No
task	Specify the task number used to check in the file (may use 'default')	No
ccmcommand	path to the ccm executable file, required if it is not on the PATH	No

7.5.1.3 Examples

```
<ccmcheckin file="c:/wa/com/foo/MyFile.java"
  comment="mycomment" />
```

Checks in the file *c:/wa/com/foo/MyFile.java*. Comment attribute *mycomment* is added as a task comment. The task used is the one set as the default.

7.5.2 CCMCheckout

7.5.2.1 Description

Task to perform a Checkout command to Continuous

7.5.2.2 Parameters

Attribute	Values	Required
file	Path to the file that the command will operate on	Yes
comment	Specify a comment.	No
task	Specify the task number used to checkin the file (may use 'default')	No
ccmcommand	path to the ccm executable file, required if it is not on the PATH	No

7.5.2.3 Examples

```
<ccmcheckout file="c:/wa/com/foo/MyFile.java"
  comment="mycomment"/>
```

Check out the file `c:/wa/com/foo/MyFile.java`. Comment attribute `mycomment` is added as a task comment. The used task is the one set as the default.

7.5.3 CCMCheckinTask

7.5.3.1 Description

Task to perform a check in default task command to Continuous

7.5.3.2 Parameters

Attribute	Values	Required
comment	Specify a comment.	No
task	Specify the task number used to check in the file (may use 'default')	No
ccmcommand	path to the ccm executable file, required if it is not on the PATH	No

7.5.3.3 Examples

```
<ccmcheckintask comment="blahblah"/>
```

Does a Checkin default task on all the checked out files in the current task.

7.5.4 CCMReconfigure

7.5.4.1 Description

Task to perform an reconfigure command to Continuous.

7.5.4.2 Parameters

Attribute	Values	Required
recurse	recurse on subproject (default false)	No
verbose	do a verbose reconfigure operation (default false)	No
ccmproject	Specifies the ccm project on which the operation is applied.	Yes
ccmcommand	path to the ccm executable file, required if it is not on the PATH	No

7.5.4.3 Examples

```
<ccmreconfigure ccmproject="ANTCCM_TEST#BMO_1"
  verbose="true" />
```

Does a Continuous *reconfigure* on the project *ANTCCM_TEST#BMO_1*.

7.5.5 CCMCreateTask

7.5.5.1 Description

Create a Continuous task.

7.5.5.2 Parameters

Attribute	Values	Required
comment	Specify a comment.	No
platform	Specify the target platform	No
ccmcommand	path to the ccm executable file, required if it is not on the PATH	No
resolver	Specify the resolver	No
release	Specify the CCM release	No
subsystem	Specify the subsystem	No
task	Specify the task number used to checkin the file (may use 'default')	No

7.5.5.3 Examples

```
<ccmcreatetask resolver="${user.name}"
  release="ANTCCM_TEST" comment="blahblah" />
```

Creates a task for the release *ANTCCM_TEST* with the current user as the resolver for this task.

7.6 Depend

A task to manage Java class file dependencies.

7.6.1 Description

The depend task works by determining which classes are out of date with respect to their source and then removing the class files of any other classes which depend on the out-of-date classes.

To determine the class dependencies, the depend task analyses the class files of all class files passed to it. Depend does not parse your source code in any way but relies upon the class references encoded into the class files by the compiler. This is generally faster than parsing the Java source.

To learn more about how this information is obtained from the class files, please refer to the Java Virtual Machine Specification (<http://java.sun.com/docs/books/vmspec/>)

Since a class' dependencies only change when the class itself changes, the depend task is able to cache dependency information. Only those class files which have changed will have their dependency information re-analysed. Note that if you change a class' dependencies by changing the source, it will be recompiled anyway. You can examine the dependency files created to understand the dependencies of your classes. Please do not rely, however, on the format of the information, as it may change in a later release.

Once depend discovers all of the class dependencies, it "inverts" this relation to determine, for each class, which other classes are dependent upon it. This "affects" list is used to discover which classes are invalidated by the out of date class. The class files of the invalidated classes are removed, triggering the compilation of the affected classes.

The depend task supports an attribute, "closure" which controls whether depend will only consider direct class-class relationships or whether it will also consider transitive, indirect relationships. For example, say there are three classes, A, which depends on B, which in-turn depend on C. Now say that class C is out of date. Without closure, only class B would be removed by depend. With closure set, class A would also be removed. Normally direct relationships are sufficient - it is unusual for a class to depend on another without having a direct relationship. With closure set, you will notice that depend typically removes far more class files.

The classpath attribute for `<depend>` is optional. If it is present, depend will check class dependencies against classes and jars on this classpath. Any classes which depend on an element from this classpath and which are older than that element will be deleted. A typical example where you would use this facility would be where you are building a utility jar and want to make sure classes which are out of date with respect to this jar are rebuilt. You should **not** include jars in this classpath which you do not expect to change, such as the JDK runtime jar or third party jars, since doing so will just slow down the dependency check. This means that if you do use a classpath for the depend task it may be different from the classpath necessary to actually compile your code.

7.6.2 Performance

The performance of the depend task is dependent on a number of factors such as class relationship complexity and how many class files are out of date. The decision about whether it is cheaper to just recompile all classes or to use the depend task will depend on the size of your project and how interrelated your classes are.

7.6.3 Limitations

There are some source dependencies which depend will not detect.

- If the Java compiler optimizes away a class relationship, there can be a source dependency without a class dependency.
- Non public classes cause two problems. Firstly depend cannot relate the class file to a source file. In the future this may be addressed using the source file attribute in the classfile. Secondly, neither depend nor the compiler tasks can detect when a non public class is missing. Inner classes are handled by the depend task.

7.6.4 Parameters

Attribute	Description	Required
srcDir	This is the directory where the source exists. depend will examine this to determine which classes are out of date. If you use multiple source directories you can pass this attribute a path of source directories.	Yes
destDir	This is the root directory of the class files which will be analysed. If this is not present, the srcdir is used.	No
cache	This is a directory in which depend can store and retrieve dependency information. If this is not present, depend will not use a cache	No
closure	This attribute controls whether depend only removes classes which directly depend on out of date classes. If this is set to true, depend will traverse the class dependency graph deleting all affected classes. Defaults to false	No
dump	If true the dependency information will be written to the debug level log	No
classpath	The classpath containing jars and classes for which <depend> should also check dependencies	No

7.6.5 Nested Elements

depend's *classpath* attribute is a PATH like structure [p 13] and can also be set via a nested *classpath* element.

7.6.6 Examples

```
<depend srcdir="${java.dir}"
        destdir="${build.classes}"
        cache="depcache"
        closure="yes"/>
```

In this example classes in the `${build.classes}` directory will be removed if they depend on out-of-date classes. Classes are considered out of date with respect to the source in the `${java.dir}` directory using the same mechanism as the `javac` task. In this instance the `depend` task caches its dependency information in the `depcache` directory.

7.7 Ant EJB Tasks User Manual

by

- Paul Austin (p_d_austin@yahoo.com)
- Tim Fennell (tfenne@rcn.com)
- Martin Gee (martin.gee@icsynergy.com)
- Conor MacNeill (conor@cortexebusiness.com.au)
- Greg Nelson (greg@netscape.com)

Version @VERSION@

\$Id: ejb.html,v 1.8 2001/08/05 09:31:23 conor Exp \$

7.7.1 Table of Contents

- Introduction [p 168]
 - EJB Tasks [p 168]
-

7.7.2 Introduction

Ant provides a number of optional tasks for developing Enterprise Java Beans (EJBs) (<http://java.sun.com/products/ejb>). In general these tasks are specific to the particular vendor's EJB Server.

At present the tasks support:

- Borland (<http://www.borland.com>) Application Server 4.5
- iPlanet (<http://www.iplanet.com>) Application Server 6.0
- jboss 2.1 (<http://www.jboss.org/>) and above EJB servers
- Weblogic (<http://www.bea.com>) 4.5.1, 5.1, and 6.0 EJB servers

Over time we expect further optional tasks to support additional EJB Servers.

Ant provides a number of optional tasks for developing Enterprise Java Beans (EJBs) (<http://java.sun.com/products/ejb>). In general these tasks are specific to the particular vendor's EJB Server. At present the tasks support Weblogic (<http://www.bea.com>) 4.5.1 and 5.1 EJB servers. Over time we expect further optional tasks to support additional EJB Servers.

7.7.3 EJB Tasks

Task	Application Servers	
blgenclient [p 186]	Borland Application Server 4.5	
ddcreator [p 169]	Weblogic 4.5.1	
ejbc [p 170]	Weblogic 4.5.1	
iplanet-ejbc [p 171]	iPlanet Application Server 6.0	
ejbjar [p 175]	Nested Elements	
	borland [p 187]	Borland Application Server 4.5
	jBoss [p 178]	jBoss
	iPlanet [p 183]	iPlanet Application Server 6.0
	weblogic [p 179]	Weblogic 5.1 & 6.0
wlrun [p 172]	Weblogic 4.5.1, 5.1 & 6.0	
wlstop [p 174]	Weblogic 4.5.1, 5.1 & 6.0	

7.7.4 ddcreator

7.7.4.1 Description:

ddcreator will compile a set of Weblogic text-based deployment descriptors into a serialized EJB deployment descriptor. The selection of which of the text-based descriptors are to be compiled is based on the standard Ant include and exclude selection mechanisms.

7.7.4.2 Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which descriptors are selected.	Yes
dest	The directory where the serialised deployment descriptors will be written	Yes
classpath	This is the classpath to use to run the underlying weblogic ddcreator tool. This must include the <code>weblogic.ejb.utils.DDCreator</code> class	No

7.7.4.3 Examples

```
<ddcreator descriptors="${dd.dir}"
  dest="${gen.classes}"
  classpath="${descriptorbuild.classpath}">
  <include name="*.txt" />
</ddcreator>
```

7.7.5 ejbc

7.7.5.1 Description:

The ejbc task will run Weblogic's ejbc tool. This tool will take a serialised deployment descriptor, examine the various EJB interfaces and bean classes and then generate the required support classes necessary to deploy the bean in a Weblogic EJB container. This will include the RMI stubs and skeletons as well as the classes which implement the bean's home and remote interfaces.

The ant task which runs this tool is able to compile several beans in a single operation. The beans to be compiled are selected by including their serialised deployment descriptors. The standard ant `include` and `exclude` constructs can be used to select the deployment descriptors to be included.

Each descriptor is examined to determine whether the generated classes are out of date and need to be regenerated. The deployment descriptor is de-serialized to discover the home, remote and implementation classes. The corresponding source files are determined and checked to see their modification times. These times and the modification time of the serialised descriptor itself are compared with the modification time of the generated classes. If the generated classes are not present or are out of date, the ejbc tool is run to generate new versions.

7.7.5.2 Parameters:

Attribute	Description	Required
descriptors	This is the base directory from which the serialised deployment descriptors are selected.	Yes
dest	The base directory where the generated classes, RIM stubs and RMI skeletons are written	Yes
manifest	The name of a manifest file to be written. This manifest will contain an entry for each EJB processed	Yes
src	The base directory of the source tree containing the source files of the home interface, remote interface and bean implementation classes.	Yes
classpath	This classpath must include both the <code>weblogic.ejbc</code> class and the classfiles of the bean, home interface, remote interface, etc of the bean being processed.	No

7.7.5.3 Examples

```
<ejbc descriptors="${gen.classes}"
      src="${src.dir}"
      dest="${gen.classes}"
      manifest="${build.manifest}"
      classpath="${descriptorbuild.classpath}">
  <include name="*.ser" />
</ejbc>
```

7.7.6 iplanet-ejbc

7.7.6.1 Description:

Task to compile EJB stubs and skeletons for the iPlanet Application Server 6.0. Given a standard EJB 1.1 XML descriptor as well as an iAS-specific EJB descriptor, this task will generate the stubs and skeletons required to deploy the EJB to iAS. Since the XML descriptors can include multiple EJBs, this is a convenient way of specifying many EJBs in a single Ant task.

For each EJB specified, the task will locate the three classes that comprise the EJB in the destination directory. If these class files cannot be located in the destination directory, the task will fail. The task will also attempt to locate the EJB stubs and skeletons in this directory. If found, the timestamps on the stubs and skeletons will be checked to ensure they are up to date. Only if these files cannot be found or if they are out of date will the iAS ejbc utility be called to generate new stubs and skeletons.

7.7.6.2 Parameters:

Attribute	Description	Required
ejbdescriptor	Standard EJB 1.1 XML descriptor (typically titled "ejb-jar.xml").	Yes
iasdescriptor	iAS-specific EJB XML descriptor (typically titled "ias-ejb-jar.xml").	Yes
dest	The is the base directory where the RMI stubs and skeletons are written. In addition, the class files for each bean (home interface, remote interface, and EJB implementation) must be found in this directory.	Yes
classpath	The classpath used when generating EJB stubs and skeletons. If omitted, the classpath specified when Ant was started will be used. Nested "classpath" elements may also be used.	No
keepgenerated	Indicates whether or not the Java source files which are generated by ejbc will be saved or automatically deleted. If "yes", the source files will be retained. If omitted, it defaults to "no".	No
debug	Indicates whether or not the ejbc utility should log additional debugging statements to the standard output. If "yes", the additional debugging statements will be generated. If omitted, it defaults to "no".	No
iashome	May be used to specify the "home" directory for this iAS installation. This is used to find the ejbc utility if it isn't included in the user's system path. If specified, it should refer to the "[install-location]/iplanet/ias6/ias" directory. If omitted, the ejbc utility must be on the user's system path.	No

7.7.6.3 Examples

```
<iplanet-ejbc ejbdescriptor="ejb-jar.xml"
               iasdescriptor="ias-ejb-jar.xml"
               dest="${build.classesdir}"
               classpath="${ias.ejbc.cpath}" />
```

```
<iplanet-ejbc ejbdescriptor="ejb-jar.xml"
               iasdescriptor="ias-ejb-jar.xml"
```

```

        dest="${build.classesdir}"
        keepgenerated="yes"
        debug="yes"
        iashome="${ias.home}" >
        <classpath>
            <pathelement path="." />
            <pathelement path="${build.classpath}" />
        </classpath>
    </iplanet-ejbc>

```

7.7.7 wlrn

7.7.7.1 Description:

The `wlrn` task is used to start a weblogic server. The task runs a weblogic instance in a separate Java Virtual Machine. A number of parameters are used to control the operation of the weblogic instance. Note that the task, and hence ant, will not complete until the weblogic instance is stopped.

7.7.7.2 Parameters:

Attribute	Description	Required for 4.5.1 and 5.1	Required for 6.0
BEA Home	The location of the BEA Home where the server's config is defined. If this attribute is present, wlrn assumes that the server will be running under Weblogic 6.0	N/A	Yes
home	The location of the weblogic home that is to be used. This is the location where weblogic is installed.	Yes	Yes. Note this is the absolute location, not relative to BEA home.
Domain	The domain to which the server belongs.	N/A	Yes
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Server. Prior to Weblogic 6.0, this is typically set to the Weblogic boot classpath. Under Weblogic 6.0 this should include all the weblogic jars	Yes	Yes
wlclasspath	The weblogic classpath used by the Weblogic Server.	No	N/A
properties	The name of the server's properties file within the weblogic home directory used to control the weblogic instance.	Yes	N/A
name	The name of the weblogic server within the weblogic home which is to be run. This defaults to "myserver"	No	No
policy	The name of the security policy file within the weblogic home directory that is to be used. If not specified, the default policy file <code>weblogic.policy</code> is used.	No	No
username	The management username used to manage the server	N/A	No
password	The server's management password	N/A	Yes
pkPassword	The private key password so the server can decrypt the SSL private key file	N/A	No
jvmargs	Additional argument string passed to the Java Virtual Machine used to run the Weblogic instance.	No	No
args	Additional argument string passed to the Weblogic instance.	No	No

7.7.7.3 Nested Elements

The wlrn task supports nested `<classpath>` and `<wlclasspath>` elements to set the respective classpaths.

7.7.7.4 Examples

This example shows the use of wlrn to run a server under Weblogic 5.1

```
<wlrn taskname="myserver"
      classpath="${weblogic.boot.classpath}"
      wlcclasspath="${weblogic.classes}:${code.jars}"
      name="myserver"
      home="${weblogic.home}"
      properties="myserver/myserver.properties"/>
```

This example shows wlrn being used to run the petstore server under Weblogic 6.0

```
<wlrn taskname="petstore"
      classpath="${weblogic.classes}"
      name="petstoreServer"
      domain="petstore"
      home="${weblogic.home}"
      password="petstorePassword"
      beahome="${bea.home}"/>
```

7.7.8 wlstop

7.7.8.1 Description:

The wlstop task is used to stop a weblogic instance which is currently running. To shut down an instance you must supply both a username and a password. These will be stored in the clear in the build script used to stop the instance. For security reasons, this task is therefore only appropriate in a development environment.

This task works for most version of Weblogic, including 6.0. You need to specify the BEA Home to have this task work correctly under 6.0

7.7.8.2 Parameters:

Attribute	Description	Required
BEAHome	This attribute selects Weblogic 6.0 shutdown.	No
classpath	The classpath to be used with the Java Virtual Machine that runs the Weblogic Shutdown comment.	Yes
user	The username of the account which will be used to shutdown the server	Yes
password	The password for the account specified in the user parameter.	Yes
url	The URL which describes the port to which the server is listening for T3 connections. For example, t3://localhost:7001	Yes
delay	The delay in seconds after which the server will stop. This defaults to an immediate shutdown.	No

7.7.8.3 Nested Element

The classpath of the `welstop` task can be set by a `<classpath>` nested element.

7.7.8.4 Examples

This example show the shutdown for a Weblogic 6.0 server

```
<wlstop classpath="${weblogic.classes}"
        user="system"
        url="t3://localhost:7001"
        password="foobar"
        beahome="${bea.home}"/>
```

7.7.9 *ejbjar*

7.7.9.1 Description:

This task is designed to support building of EJB1.1 jar files. Support is currently provided for 'vanilla' EJB1.1 jar files - i.e. those containing only the user generated class files and the standard deployment descriptor. Nested elements provide support for vendor specific deployment tools. These currently include:

- Borland Application Server 4.5
- iPlanet Application Server 6.0
- Jboss 2.1 and above
- Weblogic 5.1/6.0 session/entity beans using the `weblogic.ejbc` tool
- TOPLink for WebLogic 2.5.1-enabled entity beans

This task supports two approaches to creating `ejb` jar files. The first approach assumes a particular naming convention for deployment descriptor files. For an Account bean, for example, the deployment descriptor would be named `Account-ejb-jar.xml`. This naming convention allows the task to distinguish deployment descriptors without relying on their positioning within a source tree. It is also used to derive the name of the `.jar` file which is generated. For the example this would be `Account.jar`. Vendor specific files are assumed to be named in a similar fashion. The deployment descriptor file which defines additional weblogic specific information for the above bean would be `Account-weblogic-ejb-jar.xml`. The second approach does not require a naming convention. This approach uses a specified a jar name for the resultant `ejb` jar. If the jar name is present, then no naming convention is required. If the jar name is not specified, then the default naming convention is expected for the deployment descriptor files.

The task works as a directory scanning task, and performs an action for each deployment descriptor found. As such the includes and excludes should be set to ensure that all desired EJB1.1 descriptors are found, but no application server descriptors are found. For each descriptor found, `ejbjar` will parse the deployment descriptor to determine the necessary class files which implement the bean. These files are assembled along with the deployment descriptors into a well formed EJB jar file. Any support files which need to be included in the generated jar can be added with the `<support>` nested element. For each class included in

the jar, ejbjar will scan for any super classes or super interfaces. These will be added to the generated jar.

If no nested vendor-specific deployment elements are present, the task will simply generate a generic EJB jar. Such jars are typically used as the input to vendor-specific deployment tools. For each nested deployment element, a vendor specific deployment tool is run to generate a jar file ready for deployment in that vendor's EJB container.

The jar files are only built if they are out of date. Each deployment tool element will examine its target jar file and determine if it is out of date with respect to the class files and deployment descriptors that make up the bean. If any of these files are newer than the jar file the jar will be rebuilt otherwise a message is logged that the jar file is up to date.

7.7.9.2 Parameters:

Attribute	Description	Required
descriptorDir	The base directory under which to scan for EJB deployment descriptors. If this attribute is not specified, then the deployment descriptors must be located in the directory specified by the 'srcdir' attribute.	No
srcdir	The base directory containing the .class files that make up the bean. Note that this can be the same as the descriptorDir if all files are in the same directory tree.	Yes
destDir	The base directory into which generated jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptorDir namespace. Note that this attribute is only used if the task is generating generic jars (i.e. no vendor-specific deployment elements have been specified).	Yes
baseJarName	The base name that is used for the generated jar files. If this attribute is specified, the generic jar file name will use this value as the prefix (followed by the value specified in the 'genericJarSuffix' attribute) and the resultant ejb jar file (followed by any suffix specified in the nested element).	No
baseNameTerminator	String value used to substring out a string from the name of each deployment descriptor found, which is then used to locate related deployment descriptors (e.g. the WebLogic descriptors). For example, a basename of '.' and a deployment descriptor called 'FooBean.ejb-jar.xml' would result in a basename of 'FooBean' which would then be used to find FooBean.weblogic-ejb-jar.xml and FooBean.weblogic-cmp-rdbms-jar.xml, as well as to create the filenames of the jar files as FooBean-generic.jar and FooBean-wl.jar. This attribute is not used if the 'baseJarName' attribute is specified.	No, defaults to '-'.
genericJarSuffix	String value appended to the basename of the deployment descriptor to create the filename of the generic EJB jar file.	No, defaults to '-generic.jar'.
classpath	This classpath is used when resolving classes which are to be added to the jar. Typically nested deployment tool elements will also support a classpath which will be combined with this classpath when resolving classes	No.
flatDestDir	Set this attribute to true if you want all generated jars to be placed in the root of the destDir, rather than according to the location of the deployment descriptor within the descriptor dir hierarchy.	No.

7.7.9.3 Nested Elements

In addition to the vendor specific nested elements, the ejbjar task provides three nested elements.

7.7.9.3.1 Classpath

The `<classpath>` nested element allows the classpath to be set. It is useful when setting the classpath from a reference path. In all other respects the behaviour is the same as the classpath attribute.

7.7.9.3.2 dtd

The `<dtd>` element is used to specify the local location of DTDs to be used when parsing the EJB deployment descriptor. Using a local DTD is much faster than loading the DTD across the net. If you are running ejbjar behind a firewall you may not even be able to access the remote DTD. The supported vendor-specific nested elements know the location of the required DTDs within the vendor class hierarchy and, in general, this means `<dtd>` elements are not required. It does mean, however, that the vendor's class hierarchy must be available in the classpath when Ant is started. If you want to run Ant without requiring the vendor classes in the classpath, you would need to use a `<dtd>` element.

Attribute	Description	Required
publicId	The public Id of the DTD for which the location is being provided	Yes
location	The location of the local copy of the DTD. This can either be a file or a resource loadable from the classpath.	Yes

7.7.9.3.3 support

The `<support>` nested element is used to supply additional classes (files) to be included in the generated jars. The `<support>` element is a FileSet, so it can either reference a fileset declared elsewhere or it can be defined in-place with the appropriate `<include>` and `<exclude>` nested elements. The files in the support fileset are added into the generated EJB jar in the same relative location as their location within the support fileset. Note that when ejbjar generates more than one jar file, the support files are added to each one.

7.7.9.4 Vendor-specific deployment elements

Each vendor-specific nested element controls the generation of a deployable jar specific to that vendor's EJB container. The parameters for each supported deployment element are detailed here.

7.7.9.5 Jboss element

The jboss element searches for the jboss specific deployment descriptors and adds them to the final ejb jar file. Jboss has two deployment descriptors jboss.xml and jaws.xml (for container manager persistence only). The Jboss server uses hot deployment and does not require compilation of additional stubs and skeletons.

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
genericjarsuffix	A generic jar is generated as an intermediate step in build the weblogic deployment jar. The suffix used to generate the generic jar file is not particularly important unless it is desired to keep the generic jar file. It should not, however, be the same as the suffix setting.	No, defaults to '-generic.jar'.
suffix	String value appended to the basename of the deployment descriptor to create the filename of the WebLogic EJB jar file.	No, defaults to '.jar'.
keepgeneric	This controls whether the generic file used as input to ejbc is retained.	No, defaults to false

7.7.9.6 Weblogic element

The weblogic element is used to control the weblogic.ejbc compiler for generating weblogic EJB jars. Prior to Ant 1.3, the method of locating CMP descriptors was to use the ejbjar naming convention. So if your ejb-jar was called, Customer-ejb-jar.xml, your weblogic descriptor was called Customer-weblogic-ejb-jar.xml and your CMP descriptor had to be Customer-weblogic-cmp-rdbms-jar.xml. In addition, the <type-storage> element in the weblogic descriptor had to be set to the standard name META-INF/weblogic-cmp-rdbms-jar.xml, as that is where the CMP descriptor was mapped to in the generated jar.

There are a few problems with this scheme. It does not allow for more than one CMP descriptor to be defined in a jar and it is not compatible with the deployment descriptors generated by some tools.

In Ant 1.3, ejbjar parses the weblogic deployment descriptor to discover the CMP descriptors, which are then included automatically. This behaviour is controlled by the newCMP attribute. Note that if you move to the new method of determining CMP descriptors, you will need to update your weblogic deployment descriptor's <type-storage> element. In the above example, you would define this as META-INF/Customer-weblogic-cmp-rdbms-jar.xml.

Attribute	Description	Required
destdir	The base directory into which the generated weblogic ready jar files are deposited. Jar files are deposited in directories corresponding to their location within the descriptordir namespace.	Yes
genericjarsuffix	A generic jar is generated as an intermediate step in build the weblogic deployment jar. The suffix used to generate the generic jar file is not particularly important unless it is desired to keep the generic jar file. It should not, however, be the same as the suffix setting.	No, defaults to '-generic.jar'.
suffix	String value appended to the basename of the deployment descriptor to create the filename of the WebLogic EJB jar file.	No, defaults to '.jar'.

classpath	The classpath to be used when running the weblogic ejbc tool. Note that this tool typically requires the classes that make up the bean to be available on the classpath. Currently, however, this will cause the ejbc tool to be run in a separate VM	No
wlclasspath	Weblogic 6.0 will give a warning if the home and remote interfaces of a bean are on the system classpath used to run weblogic.ejbc. In that case, the standard weblogic classes should be set with this attribute (or equivalent nested element) and the home and remote interfaces located with the standard classpath attribute	No
keepgeneric	This controls whether the generic file used as input to ejbc is retained.	No, defaults to false
compiler	This allows for the selection of a different compiler to be used for the compilation of the generated Java files. This could be set, for example, to Jikes to compile with the Jikes compiler. If this is not set and the <code>build.compiler</code> property is set to jikes, the Jikes compiler will be used. If this is not desired, the value "default" may be given to use the default compiler	No
rebuild	This flag controls whether weblogic.ejbc is always invoked to build the jar file. In certain circumstances, such as when only a bean class has been changed, the jar can be generated by merely replacing the changed classes and not rerunning ejbc. Setting this to false will reduce the time to run ejbjar.	No, defaults to true.
keepgenerated	Controls whether weblogic will keep the generated Java files used to build the class files added to the jar. This can be useful when debugging	No, defaults to false.
args	Any additional arguments to be passed to the weblogic.ejbc tool.	No.
weblogicdtd	Deprecated. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <code><dtd></code> element, described above. If you do choose to use an attribute, you should use the <code>ejbdtd</code> attribute in preference to this one, anyway.	No.
wldtd	Deprecated. Defines the location of the weblogic-ejb-jar DTD which covers the Weblogic specific deployment descriptors. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <code><dtd></code> element, described above.	No.
ejbdtd	Deprecated. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <code><dtd></code> element, described above.	No.
newCMP	If this is set to true, the new method for locating CMP descriptors will be used.	No. Defaults to false
oldCMP	Deprecated This is an antonym for newCMP which should be used instead.	No.

noEJBC	If this attribute is set to true, Weblogic's ejbc will not be run on the EJB jar. Use this if you prefer to run ejbc at deployment time.	No.
--------	--	-----

The weblogic nested element itself supports two nested elements `<classpath>` and `<wlclasspath>` which are used to set the respective classpaths. These nested elements are useful when setting up class paths using reference Ids.

7.7.9.7 TOPLink for Weblogic element

The TopLink element is used to handle beans which use Toplink for the CMP operations. It is derived from the standard weblogic element so it supports the same set of attributes please these additional attributes

Attribute	Description	Required
toplinkdescriptor	This specifies the name of the TOPLink deployment descriptor file contained in the 'descriptor' directory.	Yes
toplinkdtd	This specifies the location of the TOPLink DTD file. This can be a file path or a file URL. This attribute is not required, but using a local DTD is recommended.	No, defaults to dtd file at www.objectpeople.com .

7.7.9.8 Examples

This example shows ejbjar being used to generate deployment jars using a Weblogic EJB container. This example requires the naming standard to be used for the deployment descriptors. Using this format will create a ejb jar file for each variation of `'*-ejb-jar.xml'` that is found in the deployment descriptor directory.

```
<ejbjar srcdir="${build.classes}"
  descriptor="${descriptor.dir}">
  <weblogic destdir="${deploymentjars.dir}"
    classpath="${descriptorbuild.classpath}"/>
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*weblogic*.xml"/>
</ejbjar>
```

If weblogic is not in the Ant classpath, the following example shows how to specify the location of the weblogic DTDs. This example also show the use of a nested classpath element.

```
<ejbjar descriptor="${src.dir}" srcdir="${build.classes}">
  <weblogic destdir="${deployment.webshop.dir}"
    keepgeneric="true"
    args="-g -keepgenerated ${ejbc.compiler}"
    suffix=".jar"
    oldCMP="false">
    <classpath>
      <pathelement path="${descriptorbuild.classpath}"/>
    </classpath>
  </weblogic>
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*-weblogic-ejb-jar.xml"/>
```

```

<dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
      location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/ejb-jar.dtd"/>
<dtd publicId="-//BEA Systems, Inc.//DTD WebLogic 5.1.0 EJB//EN"
      location="${weblogic.home}/classes/weblogic/ejb/deployment/xml/weblogic-ejb-jar.dtd"/>
</ejbjar>

```

This example shows ejbjar being used to generate a single deployment jar using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one ejb jar file - 'TheEJBJar.jar'.

```

<ejbjar srcdir="${build.classes}"
      descriptor="descriptor.dir"
      basejarname="TheEJBJar">
  <weblogic destdir="${deploymentjars.dir}"
    classpath="${descriptorbuild.classpath}"/>
  <include name="**/ejb-jar.xml"/>
  <exclude name="**/weblogic*.xml"/>
</ejbjar>

```

This example shows ejbjar being used to generate deployment jars for a TOPLink-enabled entity bean using a Weblogic EJB container. This example does not require the deployment descriptors to use the naming standard. This will create only one TOPLink-enabled ejb jar file - 'Address.jar'.

```

<ejbjar srcdir="${build.dir}"
      destdir="${solant.ejb.dir}"
      descriptor="descriptor.dir"
      basejarname="Address">
  <weblogictoplink destdir="${solant.ejb.dir}"
    classpath="${java.class.path}"
    keepgeneric="false"
    toplinkdescriptor="Address.xml"
    toplinkdtd="file:///dtdfiles/toplink-cmp_2_5_1.dtd"
    suffix=".jar"/>
  <include name="**/ejb-jar.xml"/>
  <exclude name="**/weblogic-ejb-jar.xml"/>
</ejbjar>

```

This final example shows how you would set-up ejbjar under Weblogic 6.0. It also shows the use of the <support> element to add support files

```

<ejbjar descriptor="dd.dir" srcdir="${build.classes.server}">
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*-weblogic-ejb-jar.xml"/>
  <support dir="${build.classes.server}">
    <include name="**/*.class"/>
  </support>
  <weblogic destdir="${deployment.dir}"
    keepgeneric="true"
    suffix=".jar"
    rebuild="false">
    <classpath>
      <pathelement path="${build.classes.server}"/>
    </classpath>
    <wlclasspath>

```

```

        <pathelement path="${weblogic.classes}"/>
    </wlcclasspath>
</weblogic>
</ejbjar>

```

7.7.9.9 iPlanet Application Server (iAS) element

The `<iplanet>` nested element is used to build iAS-specific stubs and skeletons and construct a JAR file which may be deployed to the iPlanet Application Server 6.0. The build process will always determine if the EJB stubs/skeletons and the EJB-JAR file are up to date, and it will do the minimum amount of work required.

Like the WebLogic element, a naming convention for the EJB descriptors is most commonly used to specify the name for the completed JAR file. For example, if the EJB descriptor `ejb/Account-ejb-jar.xml` is found in the descriptor directory, the `iplanet` element will search for an iAS-specific EJB descriptor file named `ejb/Account-ias-ejb-jar.xml` (if it isn't found, the task will fail) and a JAR file named `ejb/Account.jar` will be written in the destination directory. Note that when the EJB descriptors are added to the JAR file, they are automatically renamed `META-INF/ejb-jar.xml` and `META-INF/ias-ejb-jar.xml`.

Of course, this naming behavior can be modified by specifying attributes in the `ejbjar` task (for example, `basejarname`, `basenameterminator`, and `flatdestdir`) as well as the `iplanet` element (for example, `suffix`). Refer to the appropriate documentation for more details.

7.7.9.10 Parameters:

Attribute	Description	Required
<code>destdir</code>	The base directory into which the generated JAR files will be written. Each JAR file is written in directories which correspond to their location within the "descriptor" namespace.	Yes
<code>classpath</code>	The classpath used when generating EJB stubs and skeletons. If omitted, the classpath specified in the "ejbjar" parent task will be used. If specified, the classpath elements will be prepended to the classpath specified in the parent "ejbjar" task. Note that nested "classpath" elements may also be used.	No
<code>keepgenerated</code>	Indicates whether or not the Java source files which are generated by <code>ejbc</code> will be saved or automatically deleted. If "yes", the source files will be retained. If omitted, it defaults to "no".	No
<code>debug</code>	Indicates whether or not the <code>ejbc</code> utility should log additional debugging statements to the standard output. If "yes", the additional debugging statements will be generated. If omitted, it defaults to "no".	No
<code>iashome</code>	May be used to specify the "home" directory for this iAS installation. This is used to find the <code>ejbc</code> utility if it isn't included in the user's system path. If specified, it should refer to the [install-location]/iplanet/ias6/ias directory. If omitted, the <code>ejbc</code> utility must be on the user's system path.	No
<code>suffix</code>	String value appended to the JAR filename when creating each JAR. If omitted, it defaults to ".jar".	No

As noted above, the `iplanet` element supports additional `<classpath>` nested elements.

7.7.9.11 Examples

This example demonstrates the typical use of the `<iplanet>` nested element. It will name each EJB-JAR using the "basename" prepended to each standard EJB descriptor. For example, if the descriptor named "Account-ejb-jar.xml" is processed, the EJB-JAR will be named "Account.jar"

```
<ejbjar srcdir="${build.classesdir}"
  descriptordir="${src}" >
  <iplanet destdir="${assemble.ejbjar}"
    classpath="${ias.ejbc.cpath}" />
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*ias-*.xml"/>
</ejbjar>
```

This example demonstrates the use of a nested `classpath` element as well as some of the other optional attributes.

```
<ejbjar srcdir="${build.classesdir}"
  descriptordir="${src}" >
  <iplanet destdir="${assemble.ejbjar}"
    iashome="${ias.home}"
    debug="yes"
    keepgenerated="yes" >
    <classpath>
      <pathelement path="." />
      <pathelement path="${build.classpath}" />
    </classpath>
  </iplanet>
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*ias-*.xml"/>
</ejbjar>
```

This example demonstrates the use of `basejarname` attribute. In this case, the completed EJB-JAR will be named "HelloWorld.jar" If multiple EJB descriptors might be found, care must be taken to ensure that the completed JAR files don't overwrite each other.

```
<ejbjar srcdir="${build.classesdir}"
  descriptordir="${src}"
  basejarname="HelloWorld" >
  <iplanet destdir="${assemble.ejbjar}"
    classpath="${ias.ejbc.cpath}" />
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*ias-*.xml"/>
</ejbjar>
```

This example demonstrates the use of the `dtd` nested element. If the local copies of the DTDs are included in the classpath, they will be automatically referenced without the nested elements. In iAS 6.0 SP2, these local DTDs are found in the [iAS-install-directory]/APPS directory. In iAS 6.0 SP3, these local DTDs are found in the [iAS-install-directory]/dtd directory.

```
<ejbjar srcdir="${build.classesdir}"
  descriptordir="${src}">
  <iplanet destdir="${assemble.ejbjar}">
    classpath="${ias.ejbc.cpath}" />
  <include name="**/*-ejb-jar.xml"/>
  <exclude name="**/*ias-*.xml"/>
  <dtd publicId="-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
    location="${ias.home}/APPS/ejb-jar_1_1.dtd"/>
  <dtd publicId="-//Sun Microsystems, Inc.//DTD iAS Enterprise JavaBeans 1.0//EN"
    location="${ias.home}/APPS/IASEjb_jar_1_0.dtd"/>
</ejbjar>
```

7.7.10 BorlandGenerateClient

7.7.10.1 Description

The BorlandGenerateClient is a task dedicated to Borland Application Server v 4.5. It offers to generate the client jar file corresponding to an ejb jar file.

7.7.10.2 Parameters

Attribute	Description	Required
ejbjar	ejb jar file	yes
debug	If true, turn on the debug mode for each borland tools (java2iio, iastool ...) default = false	no
clientjar	client jar file name. If missing the client jar file name is build using the ejbjar file name: ejbjar = hellobean-ejb.jar => hellobean-ejbclient.jar	no
mode	choose the command launching mode. Two values: java or fork. default = java. Possibility to specify a classpath.	no

7.7.10.3 Examples

The following build.xml snippet is an example of how to use Borland element into the ejbjar task using the java mode.

```
<blgenclient ejbjar="lib/secutest-ejb.jar" clientjar="lib/client.jar" debug="true" mode="java">
  <classpath>
    <pathelement location="mymodule.jar"/>
  </classpath>
</blgenclient>
```

7.7.11 BorlandDeployTool

7.7.11.1 Description

The BorlandDeployTool is a vendor specific nested element for the Ejbjar optional task.

BorlandDeploymentTool is dedicated to the Borland Application Server 4.5. It generates and compiles the stubs and skeletons for all ejb described into the Deployment Descriptor, builds the jar file including the support files and verify whether the produced jar is valid or not.

7.7.11.2 Borland element

Attribute	Description	Required
destdir	The base directory into which the generated borland ready jar files are deposited	yes
debug	If true, turn on the debug mode for each borland tools (java2iio, iastool ...) default = false	no
verify	If true, turn on the verification at the end of the jar production (default = false)	no
verifyargs	extra parameter for verify command	no
suffix	String value appended to the basename of the deployment descriptor to create the filename of the Borland EJB jar file.	No, defaults to '-ejb.jar'.
basdtd	Deprecated. Defines the location of the weblogic-ejb-jar DTD which covers the Borland specific deployment descriptors. This should not be necessary if you have borland in your classpath. If you do not, you should use a nested <dtd> element, described above	no
ejbdtd	Deprecated. Defines the location of the ejb-jar DTD in the weblogic class hierarchy. This should not be necessary if you have weblogic in your classpath. If you do not, you should use a nested <dtd> element, described above.	no
generateclient	If true, turn on the generation of the corresponding ejbjar (default = false)	no

7.7.11.3 Examples

The following build.xml snippet is an example of how to use Borland element into the ejbjar task

```
<ejbjar srcdir="${build.classes}" basejarname="vsm" descriptor="hrmanager">
  <borland destdir="lib" verify="on" generateclient="on">
    <classpath refid="classpath" />
  </borland>
  <include name="**\ejb-jar.xml"/>
  <support dir="${build.classes}">
    <include name="demo\*.class"/>
    <include name="demo\helper\*.class"/>
  </support>
</ejbjar>
```

7.7.11 BorlandDeployTool

The borland element will generate into the lib dir an ejb jar file using the deployment descriptor placed into the \${rsc.dir}/hrmanager directory. The verify phase is turned on and the generate client phase as well.

7.7.11.4

7.8 FTP

7.8.1 *Description*

The ftp task implements a basic FTP client that can send, receive, list, delete files, and create directories. See below for descriptions and examples of how to perform each task.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies [p 5] for more information.

The ftp task makes no attempt to determine what file system syntax is required by the remote server, and defaults to Unix standards. *remotedir* must be specified in the exact syntax required by the ftp server. If the usual Unix conventions are not supported by the server, *separator* can be used to set the file separator that should be used instead.

See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

7.8.2 *Parameters*

Attribute	Description	Required
server	the address of the remote ftp server.	Yes
port	the port number of the remote ftp server. Defaults to port 21.	No
userid	the login id to use on the ftp server.	Yes
password	the login password to use on the ftp server.	Yes
remotedir	the directory to which to upload files on the ftp server.	No
action	the ftp action to perform, defaulting to "send". Currently supports "put", "get", "del", "list" and "mkdir".	No
binary	selects binary-mode ("yes") or text-mode ("no") transfers. Defaults to "yes"	No
passive	selects passive-mode ("yes") transfers. Defaults to "no"	No
verbose	displays information on each file transferred if set to "yes". Defaults to "no".	No
depends	transfers only new or changed files if set to "yes". Defaults to "no".	No
newer	a synonym for <i>depends</i> .	No
separator	sets the file separator used on the ftp server. Defaults to "/".	No
listing	the file to write results of the "list" action. Required for the "list" action, ignored otherwise.	No
ignoreNoncriticalErrors	flag which permits the task to ignore some non-fatal error codes sent by some servers during directory creation: wu-ftp in particular. Default: false	No
skipFailedTransfers	flag which enables unsuccessful file put, delete and get operations to be skipped with a warning and the remainder of the files still transferred. Default: false	No

7.8.3 Sending Files

The easiest way to describe how to send files is with a couple of examples:

```
<ftp server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com">
  <fileset dir="htdocs/manual"/>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and uploads all files in the `htdocs/manual` directory to the default directory for that user.

```

<ftp server="ftp.apache.org"
      remotedir="incoming"
      userid="anonymous"
      password="me@myorg.com"
      depends="yes"
>
  <fileset dir="htdocs/manual"/>
</ftp>

```

Logs in to `ftp.apache.org` as `anonymous` and uploads all new or changed files in the `htdocs/manual` directory to the `incoming` directory relative to the default directory for `anonymous`.

```

<ftp server="ftp.apache.org"
      port="2121"
      remotedir="/pub/incoming"
      userid="coder"
      password="java1"
      depends="yes"
      binary="no"
>
  <fileset dir="htdocs/manual">
    <include name="**/*.html"/>
  </fileset>
</ftp>

```

Logs in to `ftp.apache.org` at port 2121 as `coder` with password `java1` and uploads all new or changed HTML files in the `htdocs/manual` directory to the `/pub/incoming` directory. The files are transferred in text mode. Passive mode has been switched on to send files from behind a firewall.

```

<ftp server="ftp.nt.org"
      remotedir="c:\uploads"
      userid="coder"
      password="java1"
      separator="\n"
      verbose="yes"
>
  <fileset dir="htdocs/manual">
    <include name="**/*.html"/>
  </fileset>
</ftp>

```

Logs in to the Windows-based `ftp.nt.org` as `coder` with password `java1` and uploads all HTML files in the `htdocs/manual` directory to the `c:\uploads` directory. Progress messages are displayed as each file is uploaded.

7.8.4 Getting Files

Getting files from an FTP server works pretty much the same way as sending them does. The only difference is that the nested filesets use the `remotedir` attribute as the base directory for the files on the FTP server, and the `dir` attribute as the local directory to put the files into. The file structure from the FTP site is preserved on the local machine.

```
<ftp action="get"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com">
  <fileset dir="htdocs/manual" >
    <include name="**/*.html"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and recursively downloads all `.html` files from default directory for that user into the `htdocs/manual` directory on the local machine.

7.8.5 Deleting Files

As you've probably guessed by now, you use nested `fileset` elements to select the files to delete from the remote FTP server. Again, the filesets are relative to the remote directory, not a local directory. In fact, the `dir` attribute of the `fileset` is ignored completely.

```
<ftp action="del"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com" >
  <fileset>
    <include name="**/*.tmp"/>
  </fileset>
</ftp>
```

Logs in to `ftp.apache.org` as `anonymous` and tries to delete all `*.tmp` files from the default directory for that user. If you don't have permission to delete a file, a `BuildException` is thrown.

7.8.6 Listing Files

```
<ftp action="list"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com"
      listing="data/ftp.listing" >
  <fileset>
    <include name="**"/>
  </fileset>
</ftp>
```

This provides a file listing in `data/ftp.listing` of all the files on the FTP server relative to the default directory of the `anonymous` user. The listing is in whatever format the FTP server normally lists files.

7.8.7 Creating Directories

Note that with the `mkdir` action, the directory to create is specified using the `remotedir` attribute.

```
<ftp action="mkdir"
      server="ftp.apache.org"
      userid="anonymous"
      password="me@myorg.com"
      remotedir="some/remote/dir" />
```

This creates the directory `some/remote/dir` beneath the default root directory. As with all other actions, the directory separator character must be correct according to the desires of the FTP server.

7.9 icontract

7.9.1 *Description*

Instruments Java classes with iContract (<http://www.reliable-systems.com/tools/>) DBC preprocessor.

The task can generate a properties file for iControl (<http://home.sol.no/~hellesoy/icplus.html>), a graphical user interface that lets you turn on/off assertions. iControl generates a control file that you can refer to from this task using the controlfile attribute.

7.9.2 *Parameters*

Attribute	Description	Required
srcdir	Location of the java files.	Yes
instrumentdir	Indicates where the instrumented source files should go.	Yes
repositorydir	Indicates where the repository source files should go.	Yes
builddir	Indicates where the compiled instrumented classes should go. Defaults to the value of instrumentdir. <i>NOTE:</i> Don't use the same directory for compiled instrumented classes and uninstrumented classes. It will break the dependency checking. (Classes will not be reinstrumented if you change them).	No
repositorybuilddir	Indicates where the compiled repository classes should go. Defaults to the value of repositorydir.	No
pre	Indicates whether or not to instrument for preconditions. Defaults to <code>true</code> unless controlfile is specified, in which case it defaults to <code>false</code> .	No
post	Indicates whether or not to instrument for postconditions. Defaults to <code>true</code> unless controlfile is specified, in which case it defaults to <code>false</code> .	No
invariant	Indicates whether or not to instrument for invariants. Defaults to <code>true</code> unless controlfile is specified, in which case it defaults to <code>false</code> .	No
failthrowable	The full name of the Throwable (Exception) that should be thrown when an assertion is violated. Defaults to <code>java.lang.Error</code>	No
verbosity	Indicates the verbosity level of iContract. Any combination of <code>error*</code> , <code>warning*</code> , <code>note*</code> , <code>info*</code> , <code>progress*</code> , <code>debug*</code> (comma separated) can be used. Defaults to <code>error*</code>	No
quiet	Indicates if iContract should be quiet. Turn it off if many your classes extend uninstrumented classes and you don't want warnings about this. Defaults to <code>false</code>	No
updateicontrol	If set to true, it indicates that the properties file for iControl in the current directory should be updated (or created if it doesn't exist). Defaults to <code>false</code> .	No
controlfile	The name of the control file to pass to iContract. Consider using iControl to generate the file. Default is not to pass a file.	Only if updateicontrol=true
classdir	Indicates where compiled (uninstrumented) classes are located. This is required in order to properly update the iconcontrol.properties file, not for instrumentation.	Only if updateicontrol=true
targets	Name of the file that will be generated by this task, which lists all the classes that iContract will instrument. If specified, the file will not be deleted after execution. If not specified, a file will still be created, but it will be deleted after execution.	No

Note: iContract will use the java compiler indicated by the project's `build.compiler` property. See documentation of the Javac task for more information. Nested includes and excludes are also supported.

Example:

Note: iContract will use the java compiler indicated by the project's `build.compiler` property. See documentation of the Javac task for more information.

Nested includes and excludes can be done very much the same way as any subclass of `MatchingTask`.

Example:

```
<icontract
  srcdir="${build.src}"
  instrumentdir="${build.instrument}"
  repositorydir="${build.repository}"
  builddir="${build.instrclasses}"
  updateicontrol="true"
  classdir="${build.classes}"
  controlfile="control"
  targets="targets"
  verbosity="error*,warning*"
  quiet="true"
>
  <classpath refid="compile-classpath"/>
</icontract>
```

7.10 JavaCC

7.10.1 *Description*

Invokes the JavaCC (<http://www.metamata.com/javacc/>) compiler compiler on a grammar file.

To use the javacc task, set the *target* attribute to the name of the grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise javacc writes the generated files to the directory containing the grammar file.

This task only invokes JavaCC if the grammar file is newer than the generated Java files. javacc assumes that the Java class name of the generated parser is the same as the name of the grammar file, ignoring the .jj. If this is not the case, the javacc task will still work, but it will always generate the output files.

7.10.2 *Parameters*

Attribute	Description	Required
target	The grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated files to. If not set, the files are written to the directory containing the grammar file.	No
buildparser	Sets the BUILD_PARSER grammar option. This is a boolean option.	No
buildtokenmanager	Sets the BUILD_TOKEN_MANAGER grammar option. This is a boolean option.	No
cachetokens	Sets the CACHE_TOKENS grammar option. This is a boolean option.	No
choiceambiguitycheck	Sets the CHOICE_AMBIGUITY_CHECK grammar option. This is an integer option.	No
commontokenaction	Sets the COMMON_TOKEN_ACTION grammar option. This is a boolean option.	No
debuglookahead	Sets the DEBUG_LOOKAHEAD grammar option. This is a boolean option.	No
debugparser	Sets the DEBUG_PARSER grammar option. This is a boolean option.	No
debugtokenmanager	Sets the DEBUG_TOKEN_MANAGER grammar option. This is a boolean option.	No
errorreporting	Sets the ERROR_REPORTING grammar option. This is a boolean option.	No
forcelacheck	Sets the FORCE_LA_CHECK grammar option. This is a boolean option.	No
ignorecase	Sets the IGNORE_CASE grammar option. This is a boolean option.	No
javaunicodeescape	Sets the JAVA_UNICODE_ESCAPE grammar option. This is a boolean option.	No
lookahead	Sets the LOOKAHEAD grammar option. This is an integer option.	No
optimizetokenmanager	Sets the OPTIMIZE_TOKEN_MANAGER grammar option. This is a boolean option.	No
otherambiguitycheck	Sets the OTHER_AMBIGUITY_CHECK grammar option. This is an integer option.	No
sanitycheck	Sets the SANITY_CHECK grammar option. This is a boolean option.	No
static	Sets the STATIC grammar option. This is a boolean option.	No
unicodeinput	Sets the UNICODE_INPUT grammar option. This is a boolean option.	No
usercharstream	Sets the USER_CHAR_STREAM grammar option. This is a boolean option.	No
usertokenmanager	Sets the USER_TOKEN_MANAGER grammar option. This is a boolean option.	No

7.10.3 Example

```
<javacc
  target="src/Parser.jj"
  outputdirectory="build/src"
  javacchome="c:/program files/JavaCC"
  static="true"
/>
```

This invokes JavaCC on grammar file `src/Parser.jj`, writing the generated files to `build/src`. The grammar option `STATIC` is set to `true` when invoking JavaCC.

7.11 Javah

7.11.1 Description

Generates JNI headers from a Java class.

When this task executes, it will generate the C header and source files that are needed to implement native methods. JNI operates differently depending on whether JDK1.2

(<http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javah.html>) (or later) or pre-JDK1.2

(<http://java.sun.com/products/jdk/1.1/docs/tooldocs/win32/javah.html>) systems are used.

7.11.2 Parameters

Attribute	Description	Required
class	the fully-qualified name of the class (or classes, separated by commas)	Yes
outputFile	concatenates the resulting header or source files for all the classes listed into this file	Yes
destdir	sets the directory where javah saves the header files or the stub files.	
force	specifies that output files should always be written (JDK1.2 only)	No
old	specifies that old JDK1.0-style header files should be generated (otherwise output file contain JNI-style native method function prototypes) (JDK1.2 only)	No
stubs	generate C declarations from the Java object file (used with old)	No
verbose	causes Javah to print a message concerning the status of the generated files	No
classpath	the classpath to use.	No
bootclasspath	location of bootstrap class files.	No
extdirs	location of installed extensions.	No

Either outputFile or destdir must be supplied, but not both.

7.11.3 Examples

```
<javah destdir="c" class="org.foo.bar.Wibble" />
```

makes a JNI header of the named class, using the JDK1.2 JNI model. Assuming the directory 'c' already exists, the file `org_foo_bar_Wibble.h` is created there. If this file already exists, it is left unchanged.

```
<javah outputFile="wibble.h">
  <class name="org.foo.bar.Wibble,org.foo.bar.Bobble"/>
</javah>
```

is similar to the previous example, except the output is written to a file called `wibble.h` in the current directory.

```
<javah destdir="c" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes three header files, one for each of the classes named. Because the force option is set, these header files are always written when the Javah task is invoked, even if they already exist.

```
<javah destdir="c" verbose="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
<javah destdir="c" verbose="yes" stubs="yes" old="yes" force="yes">
  <class name="org.foo.bar.Wibble"/>
  <class name="org.foo.bar.Bobble"/>
  <class name="org.foo.bar.Tribble"/>
</javah>
```

writes the headers for the three classes using the 'old' JNI format, then writes the corresponding .c stubs. The verbose option will cause Javah to describe its progress.

7.12 JDepend

7.12.1 Description

Invokes the JDepend (<http://www.clarkware.com/software/JDepend.html>) parser.

This parser "traverses a set of Java source file directories and generates design quality metrics for each Java package". It allows to "automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to effectively manage and control package dependencies."

Source file directories are defined by nested `<sourcespath>`, see nested elements [p 202] .

Optionally, you can also set the `outputfile` name where the output is stored. By default the task writes its report to the standard output.

The task requires at least the JDepend 1.2 version.

Note: whereas the JDepend tool can be customized to exclude some packages, the current `jdepend` Ant Task does not have parameters to allow these exclusions. Read JDepend specific documentation for that purpose.

7.12.2 Parameters

Attribute	Description	Required
outputfile	The output file name. If not set, the output is printed on the standard output.	No
fork	Run the tests in a separate VM.	No, default is "off"
haltonerror	Stop the build process if an error occurs during the <code>jdepend</code> analysis.	No, default is "off"
timeout	Cancel the operation if it doesn't finish in the given time (measured in milliseconds). (Ignored if <code>fork</code> is disabled.)	No
jvm	The command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by <code>java.lang.Runtime.exec()</code> . (Ignored if <code>fork</code> is disabled.)	No, default "java"
dir	The directory to invoke the VM in. (Ignored if <code>fork</code> is disabled)	No
classpathref	the classpath to use, given as reference to a PATH defined elsewhere.	No

7.12.3 Nested Elements

`jdepend` supports two nested elements `<classpath>` and `<sourcespath>`, that represent PATH like structures [p 13] .

`<sourcespath>` is used to define the paths of the source code to analyze.

7.12.4 Examples

```
<jdepend classpathref="base.path">
  <sourcespath>
    <pathelement location="src" />
  </sourcespath>
</jdepend>
```

This invokes JDepend on the `src` directory, writing the output on the standard output. The classpath is defined using a classpath reference.

```
<jdepend outputfile="docs/jdepend.txt" fork="yes">
  <sourcespath>
    <pathelement location="src" />
  </sourcespath>
  <classpath>
    <pathelement location="classes" />
    <pathelement location="lib/jdepend.jar" />
  </classpath>
</jdepend>
```

This invokes JDepend in a separate VM on the `src` and `testsrc` directories, writing the output in the `<docs/jdepend.txt>` file. The classpath is defined using nested elements.

7.13 JJTree

7.13.1 Description

Invokes the JJTree (<http://www.metamata.com/javacc/>) preprocessor for the JavaCC compiler compiler. It inserts parse tree building actions at various places in the JavaCC source that it generates. The output of JJTree is run through JavaCC to create the parser.

To use the `jjtree` task, set the *target* attribute to the name of the `jjtree` grammar file to process. You also need to specify the directory containing the JavaCC installation using the *javacchome* attribute, so that ant can find the JavaCC classes. Optionally, you can also set the *outputdirectory* to write the generated file to a specific directory. Otherwise `jjtree` writes the generated JavaCC grammar file to the directory containing the JJTree grammar file.

This task only invokes JJTree if the grammar file is newer than the generated JavaCC file.

7.13.2 Parameters

Attribute	Description	Required
target	The <code>jjtree</code> grammar file to process.	Yes
javacchome	The directory containing the JavaCC distribution.	Yes
outputdirectory	The directory to write the generated file to. If not set, the files are written to the directory containing the grammar file.	No
buildnodefiles	Sets the <code>BUILD_NODE_FILES</code> grammar option. This is a boolean option.	No
multi	Sets the <code>MULTI</code> grammar option. This is a boolean option.	No
nodedefaultvoid	Sets the <code>NODE_DEFAULT_VOID</code> grammar option. This is a boolean option.	No
nodefactory	Sets the <code>NODE_FACTORY</code> grammar option. This is boolean option.	No
nodescopehook	Sets the <code>NODE_SCOPE_HOOK</code> grammar option. This is a boolean option.	No
nodeusesparser	Sets the <code>NODE_USES_PARSER</code> grammar option. This is a boolean option.	No
static	Sets the <code>STATIC</code> grammar option. This is a boolean option.	No
visitor	Sets the <code>VISITOR</code> grammar option. This is a boolean option.	No
nodepackage	Sets the <code>NODE_PACKAGE</code> grammar option. This is a string option.	No
visitorexception	Sets the <code>VISITOR_EXCEPTION</code> grammar option. This is a string option.	No
nodeprefix	Sets the <code>NODE_PREFIX</code> grammar option. This is a string option.	No

7.13.3 Example

```
<jjtree
  target="src/Parser.jjt"
  outputdirectory="build/src"
  javacchome="c:/program files/JavaCC"
  nodeusesparser="true"
/>
```

This invokes JJTree on grammar file `src/Parser.jjt`, writing the generated grammar file, `Parser.jj`, file to `build/src`. The grammar option `NODE_USES_PARSER` is set to `true` when invoking JJTree.

7.14 Jlink

7.14.1 Description:

Links entries from sub-builds and libraries.

The jlink task can be used to build jar and zip files, similar to the *jar* task. However, jlink provides options for controlling the way entries from input files are added to the output file. Specifically, capabilities for merging entries from multiple zip or jar files is available.

If a mergefile is specified directly (eg. at the top level of a *mergefiles* path element) *and* the mergefile ends in ".zip" or ".jar", entries in the mergefile will be merged into the outfile. A file with any other extension will be added to the output file, even if it is specified in the mergefiles element. Directories specified in either the mergefiles or addfiles element are added to the output file as you would expect: all files in subdirectories are recursively added to the output file with appropriate prefixes in the output file (without merging).

In the case where duplicate entries and/or files are found among the files to be merged or added, jlink merges or adds the first entry and ignores all subsequent entries.

jlink ignores META-INF directories in mergefiles. Users should supply their own manifest information for the output file.

It is possible to refine the set of files that are being jlinked. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile*, and *defaultexcludes* attributes on the *addfiles* and *mergefiles* nested elements. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns. The patterns are relative to the *base* directory.

7.14.2 Parameters:

Attribute	Description	Required
outfile	the path of the output file.	Yes
compress	whether or not the output should be compressed. <i>true</i> , <i>yes</i> , or <i>on</i> result in compressed output. If omitted, output will be uncompressed (inflated).	No
mergefiles	files to be merged into the output, if possible.	At least one of mergefiles or addfiles
addfiles	files to be added to the output.	

7.14.3 Examples

The following will merge the entries in mergefoo.jar and mergebar.jar into out.jar. mac.jar and pc.jar will be added as single entries to out.jar.

```
<jlink compress="false" outfile="out.jar">
  <mergefiles>
    <pathelement path="${build.dir}/mergefoo.jar"/>
    <pathelement path="${build.dir}/mergebar.jar"/>
  </mergefiles>
  <addfiles>
    <pathelement path="${build.dir}/mac.jar"/>
    <pathelement path="${build.dir}/pc.zip"/>
  </addfiles>
</jlink>
```

Suppose the file foo.jar contains two entries: bar.class and barnone/myClass.zip. Suppose the path for file foo.jar is build/tempbuild/foo.jar. The following example will provide the entry tempbuild/foo.jar in the out.jar.

```
<jlink compress="false" outfile="out.jar">
  <mergefiles>
    <pathelement path="build/tempbuild"/>
  </mergefiles>
</jlink>
```

However, the next example would result in two top-level entries in out.jar, namely bar.class and barnone/myClass.zip

```
<jlink compress="false" outfile="out.jar">
  <mergefiles>
    <pathelement path="build/tempbuild/foo.jar"/>
  </mergefiles>
</jlink>
```

7.15 JProbe

by

- Stephane Bailliez (sbailliez@imediacion.com)

7.15.1 Introduction

This task runs the tools from the JProbe suite.

For more information, visit <http://www.sitraka.com>. An evaluation version is available for download if you already don't own it.

This task has been written using JProbe Suite Server Side 3.0.

It is highly recommended to read the JProbe documentation to understand the values of the command line arguments described below. This document is less complete than the manual, it only gives the basic information and is not intended as a replacement to the manual.

7.15.2 Tasks

JPCoverage [p 208] Measure coverage of Java code.

JPCovMerge [p 212] Merge different snapshots into one.

JPCovReport [p 212] Create a report from a snapshot

7.15.3 JPCoverage

Perform code covering functions by comparing source code line execution to the program's source code as a whole.

7.15.3.1 Parameters

Attribute	Description	Required
home	The directory where is intalled JProbe.	Yes
vm	Indicates which virtual machine to run. Must be one of "jdk117", "jdk118" or "java2".If "java2" is specified, the user is also required to specify a path via <code>javahome</code> , otherwise it will check if the current executing VM is 1.2+ and use its <code>java.home</code> property to determine its location. in the registry for the default 1.2.x (Java 2) VM.	No, default to embedded VM if 1.2+
javahome	For Java 2 only, specifies the path of the jdk1.2.x virtual machine to be used. This is only valid when specifying the <code>vm="java2"</code> .	No, use only for java2
javaexe	Similar to <code>javahome</code> , except this is the path to the java executable.	No, use only for java2
applet	Run an applet. The default is false, unless the file under analysis ends with htm or html.	No, default is "false".
seedname	Seed name for the temporary snapshot files (files will be named seed.jpc, seed1.jpc, seed2.jpc, ...)	No, default to "snapshot"
exitprompt	Toggles display of the console prompt: "Press Enter to close this window." "always": Always displays the prompt. "never": Never displays the prompt. "error": Only displays prompt after an error.	No, default is "never"
finalsnapshot	Type of snapshot to send at program termination. Must be one of "none","coverage","all"	No, default to "coverage"
recordfromstart	Must be one of "coverage", "all", "none". If you want Coverage to start analyzing as soon as the program begins to run, use "all". If not, select "none".	No, default to "coverage"
warnlevel	Set warning level (0-3, where 0 is the least amount of warnings).	No, default to 0
snapshotdir	The path to the directory where snapshot files are stored. Choose a directory that is reachable by both the remote and local computers, and enter the same path on the command line and in the viewer.	No, default to current directory
workingdir	The physical path to the working directory for the VM.	No, default is current directory.
tracknatives	Test native methods. Note that testing native methods with Java 2 disables the JIT	No, default to "false".
classname	the name of the class to analyze.	Yes

7.15.3.2 Nested Elements

7.15.3.2.1 classpath

jpcoverage supports a nested `<classpath>` element, that represents a PATH like structure [p 13] .

7.15.3.2.2 jvmarg

Additional parameters may be passed to the VM via nested `<jvmarg>` attributes, for example:

```
<jpcoverage home="c:\jprobe" classname="MyClass">
  <jvmarg value="-classic"/>
  <classpath path="."/>
</jpcoverage>
```

would run the coverage on "MyClass" in classic mode VM.

`<jvmarg>` allows all attributes described in Command line arguments [p 14] .

7.15.3.2.3 arg

Parameters may be passed to the executed class via nested `<arg>` attributes, as described in Command line arguments [p 14] .

7.15.3.2.4 socket

Define a host and port to connect to if you want to do remote viewing.

Attribute	Description	Required
host	the host name/ip of the machine on which the Viewer is running	No, default to localhost
port	The port number on which you will connect to the Viewer	No, default to 4444

7.15.3.2.5 filters

Defines class/method filters based on pattern matching. The syntax is filters is similar to a fileset [p 25] .

Attribute	Description	Required
defaultexclude	As a default, the coverage exclude all classes and methods. Default filters are equivalent to <pre><filters> <exclude class="*" method="*" /> </filters></pre>	No, default to "true"

As seen above, nested elements are **include** and **exclude** with a name attribute.

Attribute	Description	Required
class	The class mask as a simple regular expression	No, default to *
method	The method mask as a simple regular expression	No, default to *

7.15.3.3 Example of filters

```
<filters>
  <include class="com.mycompany.*" method="*" />
  <exclude class="com.mycompany.MyClass" method="test*" />
</filters>
```

reports the coverage on all packages, classes and methods from `com.mycompany` except all methods starting by `test` on the class `MyClass` in the package `com.mycompany`

7.15.3.3.1 triggers

Define a number of events to use for interacting with the collection of data performed during coverage. For example you may run a whole application but only decide to collect data once it reaches a certain method and once it exits another one.

The only type of nested element is the `method` element (triggers are performed on method) and it has the following attributes:

Attribute	Description	Required
name	The name of the method(s) as a regular expression. The name is the fully qualified name on the form <code>package.className.method</code>	Yes
event	the event on the method that will trigger the action. Must be "enter" or "exit".	Yes
action	the action to execute. Must be one of "clear", "pause", "resume", "snapshot", "suspend", or "exit". They respectively clear recording, pause recording, resume recording, take a snapshot, suspend the recording and exit the program.	Yes

7.15.3.4 Example of triggers

```
<triggers>
  <method name="ClassName.*()" event="enter" action="snapshot" />
  <method name="ClassName.MethodName()" event="exit" action="exit" />
</triggers>
```

Will take a snapshot when it enters any method of the class `ClassName` and will exit the program once it exits the method `MethodName` of the same class.

7.15.4 JPCovMerge

7.15.4.1 Description

Perform the merge of several snapshots into a single one.

7.15.4.2 Parameters

Attribute	Description	Required
home	The directory where is installed JProbe.	Yes
tofile	the output filename that will be the result of the name.	Yes
verbose	Perform the merge in verbose mode giving details about the snapshot processing.	No. Default to false

jpcovmerge collects snapshots using the nested <FileSet> [p 25] element.

7.15.4.3 Example of merge

```
<jpcovreport home="c:\jprobe" tofile="merge.jpc" verbose="true">
  <fileset dir="./snapshots">
    <include name="snap*.jpc"/>
  </fileset>
</jpcovreport>
```

would run the merge in verbose mode on all snapshot files starting by snap in the directory snapshots. The resulting file will be named merge . jpc.

7.15.5 JPCovReport

7.15.5.1 Description

Generate a readable/printable report of a snapshot. Note that you will need Jakarta Oro (<http://jakarta.apache.org/oro/index.html>) in Ant classpath, to run the reference feature.

7.15.5.2 Parameters

Attribute	Description	Required
home	The directory where is intalled JProbe.	Yes
format	The format of the generated report. Must be "xml", "html" or "text"	No, default to "html"
type	The type of report to be generated. Must be "executive", "summary", "detailed" or "verydetailed"	No. Default to "detailed"
percent	A numeric value for the threshold for printing methods. Must be between 0 and 100.	No, default to 100
snapshot	The name of the snapshot file that is the source to the report.	Yes
tofile	The name of the generated output file	Yes
includesource	Include text of the source code lines. Only applies to format="xml" and type="verydetailed"	No. Defaults to "yes"

7.15.5.2.1 *sourcepath*

Path to source files can be set via nested sourcepath elements that are PATH like structures [p 13] .

7.15.5.2.2 *reference (only applies to format="xml")*

A reference is a set of classes whose coverage information will be checked against. Since Coverage is only able to give you information about loaded classes, it will only report classes that were at least used in some points in your tests, therefore you will not be able to know what classes are not exercised at all during your tests. The reference is an additional feature that will analyze the bytecode of all classes in a given classpath that match some filters and modify the XML report accordingly. In short, it will:

- remove the classes that do not exists in the reference classpath. (For example you might have in your report some helper test classes that you do not want to appear in the report, but are unable to filter without adding hundred of filters for all your classes).
- add classes that exists in the reference set and match the filters but are not reported.
- remove abstract methods that are incorrectly reported in JProbe 3.0 (should be fixed in a later SP)
- remove classes/methods that do not match the filters.

7.15.5.2.3 *classpath*

Path to the reference set of files can be set via nested classpath elements that are PATH like structures [p 13] .

7.15.5.2.4 *filters*

Nested elements are **include** and **exclude** with a **class** and **method** attribute.

Attribute	Description	Required
class	The class mask as a simple regular expression	No, default to *
method	The method mask as a simple regular expression	No, default to *

7.15.5.3 Example of report

```
<jpcovreport home="c:\jprobe" snapshot="merge.jpc" format="xml" tofile="result.xml">
  <sourcepath path="./src"/>
  <reference>
    <classpath path="./bin/classes"/>
    <filters>
      <include class="com.mycompany.*"/>
      <exclude class="com.mycompany.MyClass" method="test*" />
    </filters>
  </reference>
</jpcovreport>
```

would generate the report of the file `merge.jpc` and write it to `result.xml` using the source path `src`. As well, it will modify the `result.xml` by analyzing all classes in the `./bin/classes` that are part of the package `com.mycompany` except the method that start by `test` from the class `MyClass`.

7.15.5.4 Recommendation

If you generate your main code and your testcases in a separate directory, say `bin/classes` and `test/classes`. You should mostly end up with a reference such as:

```
<reference>
  <classpath path="./bin/classes"/>
</reference>
```

With such a reference, your XML report will be cleaned up against parasite classes from your testcases (that as a common practice, generally match the exact package structure of the class you exercise).

7.15.5.5 HTML reports

You will find in Ant etc directory a stylesheet called `coverage-frames.xsl`. This file can be used to generate a framed report a la javadoc similar to the one for JUnit. It needs either Xalan 1.2.2 or Xalan 2.x.

Xalan 1.2.2 (you must have `xalan.jar` and `bsf.jar` in your classpath)

```
<style processor="xalan" in="./reports/xml/results.xml" out="./reports/html/dummy.file"
  style="${ant.home}/etc/coverage-frames.xsl">
  <param name="output.dir" expression="'${basedir}/reports/html'"/>
</style>
```

Xalan 2.x (note the parameter without single quote)

```
<style processor="trax" in="./reports/xml/results.xml" out="./reports/html/dummy.file"
  style="${ant.home}/etc/coverage-frames.xsl">
  <param name="output.dir" expression="${basedir}/reports/html"/>
</style>
```

7.16 JUnit

7.16.1 Description

This task runs tests from the JUnit testing framework. The latest version of the framework can be found at <http://www.junit.org>. This task has been tested with JUnit 3.0 up to JUnit 3.7, it won't work with versions prior to JUnit 3.0.

Tests are defined by nested `test` or `batchtest` tags, see nested elements [p 215] .

7.16.2 Parameters

Attribute	Description	Required
printsummary	Print one line statistics for each testcase. Can take the values "on", "off" and "withOutAndErr" - "withOutAndErr" is the same as "on" but also includes the test's output (written to System.out and System.err).	No, default is "off"
fork	Run the tests in a separate VM.	No, default is "off"
haltonerror	Stop the build process if an error occurs during the test run.	No, default is "off"
errorProperty	The name of a property to set in the event of an error.	No.
haltonfailure	Stop the build process if a test fails (errors are considered failures as well).	No, default is "off"
failureProperty	The name of a property to set in the event of a failure (errors are considered failures as well).	No.
timeout	Cancel the individual tests if they don't finish in the given time (measured in milliseconds). Ignored if fork is disabled.	No
maxmemory	Max amount of memory to allocate to the forked VM (ignored if fork is disabled)	No
jvm	the command used to invoke the Java Virtual Machine, default is 'java'. The command is resolved by java.lang.Runtime.exec(). Ignored if fork is disabled.	No, default "java"
dir	The directory to invoke the VM in. (ignored if fork is disabled)	No

7.16.3 Nested Elements

`junit` supports a nested `<classpath>` element, that represents a PATH like structure [p 13] .

7.16.3.1 jvmarg

If fork is enabled, additional parameters may be passed to the new VM via nested `<jvmarg>` attributes, for example:

```
<junit fork="yes">
  <jvmarg value="-Djava.compiler=NONE"/>
  ...
</junit>
```

would run the test in a VM without JIT.

`<jvmarg>` allows all attributes described in Command line arguments [p 14] .

7.16.3.2 sysproperty

Use nested `<sysproperty>` elements to specify system properties required by the class. These properties will be made available to the VM during the execution of the test (either ANT's VM or the forked VM). The attributes for this element are the same as for environment variables [p 69] .

```
<junit fork="no">
  <sysproperty key="basedir" value="${basedir}"/>
  ...
</junit>
```

would run the test in ANT's VM and make the `basedir` property available to the test.

7.16.3.3 formatter

The results of the tests can be printed in different formats. Output will always be sent to a file unless you set the `usefile` attribute to false, the name of the file is determined by the name of the test and can be set by the `outfile` attribute of `<test>`.

There are three predefined formatters, one prints the test results in XML format, the other emit plain text. The formatter named "brief" will only print detailed information for testcases that failed, while "plain" gives a little statistics line for all test cases. Custom formatters that need to implement `org.apache.tools.ant.taskdefs.optional.junit.JUnitResultFormatter` can be specified.

Attribute	Description	Required
type	Use a predefined formatter (either "xml", "plain" or "brief").	Exactly one of these.
classname	Name of a custom formatter class.	
extension	Extension to append to the output filename.	Yes, if classname has been used.
usefile	Boolean that determines whether output should be sent to a file.	No, default true.

7.16.3.4 test

Defines a single test class.

Attribute	Description	Required
name	Name of the test class	Yes
fork	Run the tests in a separate VM. Overrides value set in <code><junit></code> .	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in <code><junit></code> .	No
errorProperty	The name of a property to set in the event of an error. Overrides value set in <code><junit></code> .	No.
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in <code><junit></code> .	No
failureProperty	The name of a property to set in the event of a failure (errors are considered failures as well). Overrides value set in <code><junit></code> .	No.
todir	Directory to write the reports to.	No, default is current directory.
outfile	Base name of the test result. The full filename is determined by this attribute and the extension of <code>formatter</code> .	No, default is <code>TEST - name</code> using the <code>name</code> attribute.
if	Only run test if the named property is set.	No
unless	Only run test if the named property is not set.	No

Tests can define their own formatters via nested `<formatter>` elements.

7.16.3.5 batchtest

Define a number of tests based on pattern matching.

`batchtest` collects the included files from any number of nested `<fileset>` [p 25] s. It then generates a test class name for each file that ends in `.java` or `.class`.

Attribute	Description	Required
fork	Run the tests in a separate VM. Overrides value set in <code><junit></code> .	No
haltonerror	Stop the build process if an error occurs during the test run. Overrides value set in <code><junit></code> .	No
errorProperty	The name of a property to set in the event of an error. Overrides value set in <code><junit></code> .	No.
haltonfailure	Stop the build process if a test fails (errors are considered failures as well). Overrides value set in <code><junit></code> .	No
failureProperty	The name of a property to set in the event of a failure (errors are considered failures as well). Overrides value set in <code><junit></code>	No.
todir	Directory to write the reports to.	No, default is current directory.
if	Only run tests if the named property is set.	No
unless	Only run tests if the named property is not set.	No

Batchtests can define their own formatters via nested `<formatter>` elements.

7.16.4 Examples

```
<junit>
  <test name="my.test.TestCase" />
</junit>
```

Runs the test defined in `my.test.TestCase` in the same VM. No output will be generated unless the test fails.

```
<junit printsummary="yes" fork="yes" haltonfailure="yes">
  <formatter type="plain" />
  <test name="my.test.TestCase" />
</junit>
```

Runs the test defined in `my.test.TestCase` in a separate VM. At the end of the test a single line summary will be printed. A detailed report of the test can be found in `TEST-my.test.TestCase.txt`. The build process will be stopped if the test fails.

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <pathelement location="${build.tests}" />
    <pathelement path="${java.class.path}" />
  </classpath>

  <formatter type="plain" />

  <test name="my.test.TestCase" haltonfailure="no" outfile="result" >
    <formatter type="xml" />
  </test>

  <batchtest fork="yes" todir="${reports.tests}">
```

```
<fileset dir="${src.tests}">
  <include name="**/*Test*.java" />
  <exclude name="**/AllTests.java" />
</fileset>
</batchtest>
</junit>
```

Runs `my.test.TestCase` in the same VM (ignoring the given CLASSPATH), only a warning is printed if this test fails. In addition to the plain text test results, for this test a XML result will be output to `result.xml`.

For each matching file in the directory `${src.tests}` a test is run in a separate VM. If a test fails, the build process is aborted. Results are collected in files named `TEST-name.txt` and written to `${reports.tests}`.

By using the `errorProperty` and `failureProperty` tags, it is possible to perform setup work (such as starting an external server), execute the test, clean up, and still fail the build in the event of a failure.

7.17 JUnitReport

Merge the individual XML files generated by the JUnit task and eventually apply a stylesheet on the resulting merged document to provide a browsable report of the testcases results.

7.17.1 Requirements

The task will run with either Xalan 2.x (<http://xml.apache.org/xalan-j/>) or Xalan 1.2.2 (http://xml.apache.org/dist/xalan-j/old/xalan-j_1_2_2.zip).

*Note: For a framed format Xalan 1.2.2 will need Xerces(xerces.jar) as well as BSF(bsf.jar) that can be found in the distribution archive. Xerces will need to be before any other parser (such as the shipped crimson). It is *highly* recommended to use Xalan2 instead because Xalan1 is no more supported.*

7.17.2 Parameters

Attribute	Description	Required
tofile	The name of the XML file that will aggregate all individual XML testsuite previously generated by the JUnit task.	No. Default to TESTS-Test-Suites.xml
todir	The directory where should be written the file resulting from the individual XML testsuite aggregation.	No. Default to current directory

7.17.3 Nested Elements

7.17.3.1 fileset

`JUnitReport` collects individual xml files generated by the JUnit task using the nested `<FileSet>` [p 25] element.

7.17.3.2 report

Generate a browsable report based on the document created by the merge.

7.17.4 Parameters

Attribute	Description	Required
format	The format of the generated report. Must be "noframes" or "frames".	No, default to "frames"
styledir	The directory where the stylesheets are defined. They must be conforming to the following conventions: • frames format: the stylesheet must be named <i>junit-frames.xsl</i>. • noframes format: the stylesheet must be named <i>junit-noframes.xsl</i>.	No. Default to embedded stylesheets.
todir	The directory where the files resulting from the transformation should be written to.	No. Default to current directory

7.17.5 Example of report

```
<junitreport todir="./reports">
  <fileset dir="./reports">
    <include name="TEST-*.xml"/>
  </fileset>
  <report format="frames" todir="./report/html"/>
</junitreport>
```

would generate a TESTS-TestSuites.xml file in the directory `reports` and generate the default framed report in the directory `report/html`.

7.18 MMetrics

by

- Stephane Bailliez (sbailliez@imediacion.com)

7.18.1 Requirements

This task requires Metamata Development environment 2.0/Webgain Quality Analyzer 2.0. An evaluation version is available at Webgain (http://www.webgain.com/products/quality_analyzer). (Though you will not be able to use Metrics from the command line if you do not have a registered version). You also need a TRaX compliant processor (such as Xalan 2.x (<http://xml.apache.org/xalan-j>)) via JAXP 1.1

7.18.2 Description

Invokes the Metamata Metrics / WebGain Quality Analyzer source code analyzer on a set of Java files.

mmetrics will compute the metrics of a set of Java files and write the results to an XML file. As a convenience, a stylesheet is given in `etc` directory, so that an HTML report can be generated from the XML file.

7.18.3 Parameters

Attribute	Description	Required
metamatahome	The home directory containing the Metamata distribution.	Yes
tofile	The XML where the resulting metrics will be written to.	Yes
granularity	Metrics granularity of the source files. Must be either <i>files</i> (compilation-units), <i>types</i> (types and compilation-units) or <i>methods</i> (methods, types and compilation-units).	Yes
maxmemory	Set the maximum memory for the JVM. this is a convenient way to set the <code>-mx</code> or <code>-Xmx</code> argument.	No

7.18.4 Nested elements

For specifying the source code to analyze, you can either use a `path` or `fileset` elements (though a single `path` element is preferred, see note below).

7.18.4.1 jvmarg

Additional parameters may be passed to the VM via nested `<jvmarg>` attributes. `<jvmarg>` allows all attributes described in Command line arguments [p 14] .

7.18.4.2 classpath

Sets class path (also source path unless one explicitly set). Overrides METAPATH/CLASSPATH environment variables. The `classpath` element represents a PATH like structure [p 13] .

7.18.4.3 sourcepath

Sets source path. Overrides the SOURCEPATH environment variable. The `sourcepath` element represents a PATH like structure [p 13] .

7.18.4.4 path

Sets the list of directories to analyze the code for metrics.;It represents a PATH structure [p 13] .

7.18.4.5 fileset

Sets a set of files to analyze for metrics.source It represents a FILESET structure [p 25] .

Note: For the sake of readability, it is highly recommended to analyze for a single unique directory instead than using filesets or several directories. Otherwise there will be multiple metrics outputs without any way to know what metrics refers to what source. Chance are also that the XML handler that does some heuristic will be confused by the different outputs.

7.18.5 Example

```
<mmetrics target="mmetrics.xml"
  metamatahome="c:/metamata"
  granularity="methods">
  <classpath>
    <pathelement location="c:/metamata/examples/metricsexamples"/>
  </classpath>
  <sourcepath>
    <pathelement location="c:/metamata/examples/metricsexamples"/>
  </sourcepath>
  <path>
    <pathelement location="c:/metamata/examples/metricsexamples"/>
  </path>
</mmetrics>
```

This invokes Metamata Metrics installed in `c:/metamata` on the metrics example. (Note that here, `classpath` and `sourcepath` are not normally not needed)

7.18.6 Generating a report

As a convenience, there is an XSL file(`mmetrics-frames.xsl`) that allows you to generate a full framed HTML report of the metrics. You can find it in the *etc* directory of Ant. As it uses the Xalan extensions, you will need BSF, Xalan and Xerces to run it. The stylesheet takes an *output.dir* parameter (otherwise it will be generated in the current directory), it can be run in Ant as follows:

```
<java classname="org.apache.xalan.xslt.Process">
  <param value="-IN"/>
  <param value="{metrics.xml}"/>
  <param value="-XSL"/>
  <param value="metrics-frames.xsl"/>
  <param value="-PARAM"/>
  <param value="output.dir"/>
  <param value="{report.dir}">"/>
</java>
```

7.19 MAudit

by

- Stephane Bailliez (sbailliez@imediacion.com)

7.19.1 Requirements

This task requires Metamata Development environment 2.0/Webgain Quality Analyzer 2.0. An evaluation version is available at Webgain (http://www.webgain.com/products/quality_analyzer). , Jakarta Oro (<http://jakarta.apache.org/oro>) and a XML parser (via JAXP).

7.19.2 Description

Invokes the Metamata Audit/ Webgain Quality Analyzer on a set of Java files.

maudit performs static analysis of the Java source code and byte code files to find and report errors of style and potential problems related to performance, maintenance and robustness. . As a convenience, a stylesheet is given in `etc` directory, so that an HTML report can be generated from the XML file.

7.19.3 Parameters

Attribute	Description	Required
tofile	The XML file to which the Audit result should be written to.	Yes
metamatahome	The home directory containing the Metamata distribution.	Yes
fix	Automatically fix certain errors (those marked as fixable in the manual).	No. Default to false.
list	Creates listing file for each audited file. A .maudit file will be generated in the same location as the source file.	No. Default to false.
unused	Finds declarations unused in search paths. It will look for unused global declarations in the source code within a use domain specified by the <code>searchpath</code> element.	No. Default to false.
maxmemory	Set the maximum memory for the JVM. this is a convenient way to set the <code>-mx</code> or <code>-Xmx</code> argument.	No

7.19.4 Nested elements

7.19.4.1 jvmarg

Additional parameters may be passed to the VM via nested `<jvmarg>` attributes. `<jvmarg>` allows all attributes described in Command line arguments [p 14] .

7.19.4.2 classpath

Sets class path (also source path unless one explicitly set). Overrides METAPATH/CLASSPATH environment variables. The `classpath` element represents a PATH like structure [p 13] .

7.19.4.3 sourcepath

Sets source path. Overrides the SOURCEPATH environment variable. The `sourcepath` element represents a PATH like structure [p 13] .

7.19.4.4 searchpath

Sets the search path to use as the use domain when looking for unused global declarations. The `searchpath` element represents a PATH like structure [p 13] .

7.19.4.5 fileset

Sets the Java files to audit via a FILESET [p 25] structure. Whatever the filter is, only the files that ends with `.java` will be included for processing. Note that the base directory used for the fileset MUST be the root of the source files otherwise package names deduced from the file path will be incorrect.

7.19.5 Example

```
<maudit tofile="c:/metamata/examples/auditexamples/audit.xml"
      metamatahome="c:/metamata" fix="yes">
  <classpath>
    <pathelement location="c:/metamata/examples/auditexamples"/>
  </classpath>
  <sourcepath>
    <pathelement location="c:/metamata/examples/auditexamples"/>
  </sourcepath>
  <fileset dir="c:/metamata/examples/auditexamples">
    <include name="*.java"/>
  </fileset>
</maudit>
```

This invokes Metamata Audit installed in `c:/metamata` on the audit examples and fix automatically the fixable errors.

7.19.6 Generating a report

As a convenience, there is an XSL file(`maudit-frames.xml`) that allows you to generate a full framed HTML report of the metrics. You can find it in the *etc* directory of Ant. As it uses the Xalan extensions, you will need BSF, Xalan and Xerces to run it. The stylesheet takes an *output.dir* parameter (otherwise it will be generated in the current directory), it can be run in Ant as follows:

```
<java classname="org.apache.xalan.xslt.Process">
  <param value="-IN"/>
  <param value="{metrics.xml}"/>
  <param value="-XSL"/>
  <param value="maudit-frames.xml"/>
  <param value="-PARAM"/>
  <param value="output.dir"/>
  <param value="&quot;'{report.dir}'&quot;"/>
</java>
```

7.20 MimeMail

7.20.1 Description

Sends SMTP mail with MIME attachments. If no attachments are needed, use the built-in Mail [p 97] task. JavaMail (<http://java.sun.com/products/javamail/index.html>) and Java Activation Framework (<http://java.sun.com/products/javabeans/glasgow/jaf.html>) are required for this task.

Multiple files can be attached using FileSets. [p 25]

7.20.2 Parameters

Attribute	Description	Required
message	The message body	No, but only one of 'message' or 'messageFile' may be specified. If not specified, a fileset must be provided.
messageFile	A filename to read and used as the message body	
messageMimeType	MIME type to use for 'message' or 'messageFile' when attached.	No, defaults to "text/plain"
tolist	Comma-separated list of To: recipients	Yes, at least one of 'tolist', 'cclist', or 'bcclist' must be specified.
cclist	Comma-separated list of CC: recipients	
bcclist	Comma-separated list of BCC: recipients	
mailhost	Host name of the mail server.	No, default to "localhost"
subject	Email subject line.	No
from	Email address of sender.	Yes
failonerror	Stop the build process if an error occurs sending the e-mail.	No, default to "true"

7.20.3 Examples

Send a single HTML file as the body of a message

```
<mimemail messageMimeType="text/html" messageFile="overview-summary.html"
  tolist="you" subject="JUnit Test Results: ${TODAY}" from="me"/>
```

Sends all files in a directory as attachments

```
<mimemail message="See attached files" tolist="you" subject="Attachments" from="me">
  <fileset dir=".">
    <include name="dist/*.*/>
  </fileset>
</mimemail>
```

7.21 MParse

by

- Stephane Bailliez (sbailliez@imediacion.com)

7.21.1 Requirements

This task requires Metamata Development environment 2.0 freely available at Metamata (<http://www.metamata.com>).

7.21.2 Description

Invokes the Metamata MParse (<http://www.metamata.com/parse.html>) compiler compiler on a grammar file.

To use the *mparse* task, set the *target* attribute to the name of the grammar file to process. You also need to specify the directory containing the Metamata installation using the *metamatahome* attribute, so that Ant can find the MParse classes.

This task only invokes MParse if the grammar file is newer than the generated Java files. MParse assumes that the Java class name of the generated parser is the same as the name of the grammar file, less the .jj extension.

For additional information about MParse, please consult the online manual available here (<http://download.metamata.com/parse.pdf>) (PDF)

7.21.3 Parameters

Attribute	Description	Required
target	The .jj grammar file to process. It will only be processed if the grammar is newer than the corresponding .java file.	Yes
metamatahome	The home directory containing the Metamata distribution.	Yes
verbose	Enable all messages	No
debugparser	Enables parser debugging	No
debugscanner	Enables scanner debugging	No
cleanup	Remove the intermediate Sun JavaCC file created during the transformation of the grammar file.	No. Default to false
maxmemory	Set the maximum memory for the JVM. this is a convenient way to set the -mx or -Xmx argument.	No

7.21.4 Nested elements

7.21.4.1 jvmarg

Additional parameters may be passed to the VM via nested `<jvmarg>` attributes. `<jvmarg>` allows all attributes described in Command line arguments [p 14] .

7.21.4.2 classpath

Sets class path (also source path unless one explicitly set). Overrides METAPATH/CLASSPATH environment variables. The `classpath` element represents a PATH like structure [p 13] .

7.21.4.3 sourcepath

Sets source path. Overrides the SOURCEPATH environment variable. The `sourcepath` element represents a PATH like structure [p 13] .

7.21.5 Example

```
<mparse target="c:/metamata/examples/parseexamples/javagrammars/singlefile/JavaParser.jj"
        metamatahome="c:/metamata" cleanup="true">
  <classpath>
    <pathelement location="c:/metamata/examples/" />
  </classpath>
</mparse>
```

This invokes Metamata MParse installed in `c:/metamata` on one of the grammar file example (`JavaParser.jj`) and cleans up the intermediate Sun JavaCC file.

7.22 Native2Ascii

7.22.1 Description:

Converts files from native encodings to ASCII with escaped Unicode. A common usage is to convert source files maintained in a native operating system encoding, to ASCII prior to compilation.

Files in the directory *src* are converted from a native encoding to ASCII. By default, all files in the directory are converted. However, conversion may be limited to selected files using *includes* and *excludes* attributes. For more information on file matching patterns, see the section on directory based tasks [p 17] . If no *encoding* is specified, the default encoding for the JVM is used. If *ext* is specified, then output files are renamed to use it as a new extension. More sophisticated file name translations can be achieved using a nested `<mapper>` element. By default an identity mapper [p 28] will be used. If *dest* and *src* point to the same directory, the *ext* attribute or a nested `<mapper>` is required.

This task forms an implicit File Set [p 25] , and supports all attributes of `<fileset>` (*dir* becomes *src*) as well as nested `<include>`, `<exclude>`, and `<patternset>` elements.

Attribute	Description	Required
reverse	Reverse the sense of the conversion, i.e. convert from ASCII to native	No
encoding	The native encoding the files are in (default is the default encoding for the JVM)	No
src	The directory to find files in (default is <i>basedir</i>)	No
dest	The directory to output file to	Yes
ext	File extension to use in renaming output files	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No

7.22.2 Examples

```
<native2ascii encoding="EUCJIS" src="srcdir" dest="srcdir"
  includes="**/*.eucjis" ext=".java" />
```

Converts all files in the directory *srcdir* ending in *.eucjis* from the EUCJIS encoding to ASCII and renames them to end in *.java*.

```
<native2ascii encoding='EUCJIS' src="native/japanese" dest="src"
  includes="**/*.java" />
```

Converts all the files ending in *.java* in the directory *native/japanese* to ASCII, placing the results in the directory *src*. The names of the files remain the same.

7.23 NetRexxC

7.23.1 Description

Compiles a NetRexx (<http://www2.hursley.ibm.com/netrexx>) source tree within the running (Ant) VM.

The source and destination directory will be recursively scanned for NetRexx source files to compile. Only NetRexx files that have no corresponding class file or where the class file is older than the java file will be compiled.

Files in the source tree are copied to the destination directory, allowing support files to be located properly in the classpath. The source files are copied because the NetRexx compiler cannot produce class files in a specific directory via parameters

The directory structure of the source tree should follow the package hierarchy.

It is possible to refine the set of files that are being compiled/copied. This can be done with the *includes*, *includesfile*, *excludes*, *excludesfile* and *defaultexcludes* attributes. With the *includes* or *includesfile* attribute you specify the files you want to have included by using patterns. The *exclude* or *excludesfile* attribute is used to specify the files you want to have excluded. This is also done with patterns. And finally with the *defaultexcludes* attribute, you can specify whether you want to use default exclusions or not. See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns.

This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `srcdir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

7.23.2 Parameters

Attribute	Description	Required
binary	Whether literals are treated as the java binary type rather than the NetRexx types	No
classpath	The classpath to use during compilation	No
comments	Whether comments are passed through to the generated java source	No
compact	Whether error messages come out in compact or verbose format	No
compile	Whether the NetRexx compiler should compile the generated java code	No
console	Whether or not messages should be displayed on the 'console'	No
crossref	Whether variable cross references are generated	No
decimal	Whether decimal arithmetic should be used for the NetRexx code	No
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No

destDir	the destination directory into which the NetRexx source files should be copied and then compiled	Yes
diag	Whether diagnostic information about the compile is generated	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
explicit	Whether variables must be declared explicitly before use	No
format	Whether the generated java code is formatted nicely or left to match NetRexx line numbers for call stack debugging	No
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
java	Whether the generated java code is produced	No
keep	Sets whether the generated java source file should be kept after compilation. The generated files will have an extension of .java.keep, not .java	No
logo	Whether the compiler text logo is displayed when compiling	No
replace	Whether the generated .java file should be replaced when compiling	No
savelog	Whether the compiler messages will be written to NetRexxC.log as well as to the console	No
sourcedir	Tells the NetRexx compiler to store the class files in the same directory as the source files. The alternative is the working directory	No
srcDir	Set the source dir to find the source NetRexx files	Yes
strictargs	Tells the NetRexx compiler that method calls always need parentheses, even if no arguments are needed, e.g. <code>aStringVar.getBytes</code> vs. <code>aStringVar.getBytes()</code>	No
strictassign	Tells the NetRexx compile that assignments must match exactly on type	No
strictcase	Specifies whether the NetRexx compiler should be case sensitive or not	No
strictimport	Whether classes need to be imported explicitly using an <code>import</code> statement. By default the NetRexx compiler will import certain packages automatically	No
strictprops	Whether local properties need to be qualified explicitly using <code>this</code>	No
strictsignal	Whether the compiler should force catching of exceptions by explicitly named types	No
symbols	Whether debug symbols should be generated into the class file	No
time	Asks the NetRexx compiler to print compilation times to the console	No
trace	Turns on or off tracing and directs the resultant trace output	No
utf8	Tells the NetRexx compiler that the source is in UTF8	No
verbose	Whether lots of warnings and error messages should be generated	Yes

7.23.3 *Examples*

```
<netrexxc srcDir="/source/project" includes="vnr/util/*"  
destDir="/source/project/build" class-  
path="/source/project2/proj.jar"  comments="true"  crossref="false"  
replace="true" keep="true"/>
```

7.24 Perforce Tasks User Manual

by

- Les Hughes (leslie.hughes@rubus.com)
- Kirk Wylie (kirk@radik.com)

Version 1.1 - 2001/01/09

7.24.1 Contents

- Introduction [p 234]
- The Tasks [p 234]
- Change History [p 240]

7.24.2 Introduction

These tasks provide an interface to the Perforce (<http://www.perforce.com>) SCM. The `org.apache.tools.ant.taskdefs.optional.perforce` package consists of a simple framework to support p4 functionality as well as some Ant tasks encapsulating frequently used (by me :-) p4 commands. However, the addition of new p4 commands is a pretty simple task (see the source). Although it is possible to use these commands on the desktop, they were primarily intended to be used by automated build systems.

Note: These tasks require the oro 2.0.XXX (<http://jakarta.apache.org/oro>) regular expression package. Simply download this package and copy the `jakarta-oro-2.0.XXX.jar` file into Ant's lib directory. You will also need the Perforce client executable (`p4` or `p4.exe` but not `p4win.exe`) in your path.

7.24.3 The Tasks

P4Sync [p 236]	Synchronise a workspace to a depot
P4Change [p 236]	Request a new changelist from the Perforce server
P4Edit [p 237]	Open files for edit (checkout)
P4Submit [p 237]	Submit a changelist to the Perforce server (checkin)
P4Have [p 237]	List current files in client view, useful for reporting
P4Label [p 238]	Create a label reflecting files in the current workspace
P4Counter [p 238]	Obtain or set the value of a counter
P4Reopen [p 239]	Move files between changelists
P4Revert [p 240]	Revert files

7.24.3.1 General P4 Properties

Each p4 task requires a number of settings, either through build-wide properties, individual attributes or environment variables. These are

Property	Attribute	Env Var	Description	Default
p4.port	port	P4PORT	The p4d server and port to connect to	perforce:1666
p4.client	client	P4CLIENT	The p4 client spec to use	The logged in username
p4.user	user	P4USER	The p4 username	The logged in username
--	view	--	The client, branch or label view to operate upon. See the p4 user guide for more info.	//...

Your local installation of Perforce may require other settings (e.g. P4PASSWD, P4CONFIG). At the moment, these can only be set outside of Ant, as environment variables.

7.24.3.2 Examples

Setting in the environment:-

(Unix csh)

```
setenv P4PORT myperforcebox:1666
```

(Unix sh et al)

```
P4USER=myp4userid; export P4USER
```

Using build properties:-

```
<property name="p4.client" value="nightlybuild"/>
```

Using task attributes:-

```
<p4whatever
  port="myserver:1666"
  client="smoketest"
  user="smoketestdude"
  .
  .
  .
/>
```

For more information regarding the underlying 'p4' commands you are referred to the Perforce Command Reference available from the Perforce website (<http://www.perforce.com/>).

7.24.3.3 Taskdefs

Standard taskdefs (for you to copy'n'paste)

```
<taskdef name="p4sync" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Sync" />
<taskdef name="p4change" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Change" />
<taskdef name="p4edit" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Edit" />
<taskdef name="p4submit" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Submit" />
<taskdef name="p4have" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Have" />
<taskdef name="p4label" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Label" />
<taskdef name="p4counter" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Counter" />
  <taskdef name="p4reopen" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Reopen" />
  <taskdef name="p4revert" classname="org.apache.tools.ant.taskdefs.optional.perforce.P4Revert" />
```

7.24.4 Task Descriptions

7.24.5 P4Sync

7.24.5.1 Description:

Synchronize the current workspace with the depot.

7.24.5.2 Parameters

Attribute	Description	Required
force	force a refresh of files	no
label	sync client to label	no

7.24.5.3 Examples

```
<p4sync label="nightlybuild-0.0123" force="yes" />
<p4sync view="//depot/projects/projectfoo/main/src/..." />
```

7.24.6 P4Change

7.24.6.1 Description:

Request a new changelist from the Perforce server. This task sets the `${p4.change}` property which can then be passed to P4Submit or P4Edit.

7.24.6.2 Parameters

Attribute	Description	Required
description	Description for ChangeList. If none specified, it will default to "AutoSubmit By Ant"	No.

7.24.6.3 Examples

```
<p4change description="Change Build Number in Script">
```

7.24.7 P4Edit

7.24.7.1 Description:

Open file(s) for edit. P4Change should be used to obtain a new changelist for P4Edit as, although P4Edit can open files to the default change, P4Submit cannot yet submit it.

7.24.7.2 Parameters

Attribute	Description	Required
view	The filespec to request to edit	Yes
change	An existing changelist number to assign files to.	No, but see above.

7.24.7.3 Examples

```
<p4edit
  view="//depot/projects/projectfoo/main/src/Blah.java..."
  change="${p4.change}" />
```

7.24.8 P4Submit

7.24.8.1 Description:

Submit a changelist, usually obtained from P4Change.

7.24.8.2 Parameters

Attribute	Description	Required
change	The changelist number to submit	Yes

7.24.8.3 Examples

```
<p4submit change="${p4.change}" />
```

7.24.9 P4Have

7.24.9.1 Description:

List handy file info reflecting the current client contents.

7.24.9.2 Parameters

Attribute	Description	Required
None	--	--

7.24.9.3 Examples

```
<p4have />
```

7.24.10 P4Label

7.24.10.1 Description:

Create a new label and set contents to reflect current client file revisions.

7.24.10.2 Parameters

Attribute	Description	Required
name	The name of the label	Yes
view	client view to use for label	No
desc	Label Description	No
lock	Lock the label once created.	No

7.24.10.3 Examples

```
<p4label
    name="NightlyBuild:${DSTAMP}:${TSTAMP}"
    desc="Auto Nightly Build"
    lock="locked"
/>
```

7.24.11 P4Counter

7.24.11.1 Description:

Obtain or set the value of a counter. When used in its base form (where only the counter name is provided), the counter value will be printed to the output stream. When the value is provided, the counter will be set to the value provided. When a property name is provided, the property will be filled with the value of the counter. You may not specify to both get and set the value of the counter in the same Task.

The user performing this task must have Perforce "review" permissions as defined by Perforce protections in order for this task to succeed.

7.24.11.2 Parameters

Attribute	Description	Required
name	The name of the counter	Yes
value	The new value for the counter	No
property	The property to be set with the value of the counter	No

7.24.11.3 Examples

Print the value of the counter named "last-clean-build" to the output stream:

```
<p4counter name="last-clean-build" />
```

Set the value of the counter based on the value of the "TSTAMP" property:

```
<p4counter name="last-clean-build" value="${TSTAMP}" />
```

Set the value of the "p4.last.clean.build" property to the current value of the "last-clean-build" counter:

```
<p4counter name="last-clean-build" property="${p4.last.clean.build}" />
```

7.24.12 P4Reopen

7.24.12.1 Description:

Move (or reopen in Perforce speak) checkout files between changelists.

7.24.12.2 Parameters

Attribute	Description	Required
tochange	The changelist to move files to.	Yes

7.24.12.3 Examples

Move all open files to the default changelist

```
<p4reopen view="//..." tochange="default" />
```

Create a new changelist then reopen into it, any files from the view //projects/foo/main/...

```
<p4change description="Move files out of the way" />
<p4reopen view="//projects/foo/main/..." tochange="${p4.change}" />
```

7.24.13 P4Revert

7.24.13.1 Description:

Reverts files.

7.24.13.2 Parameters

Attribute	Description	Required
change	The changelist to revert.	No
revertOnlyUnchanged	Revert only unchanged files (p4 revert -a)	No

7.24.13.3 Examples

Revert everything!

```
<p4revert view="//..." />
```

Revert any unchanged files in the default change

```
<p4revert change="default" revertonlyunchanged="true" />
```

7.24.14 Change History

Sept 2000	--	Internal Release within Rubus
Nov 2000	V1.0	Initial Release donated to ASF :-)
Jan 2001	V1.1	fixed cross platform (NT/Unix) bug refactored p4 output handling code refactored exec'ing code

7.25 Ant PropertyFile Task User Manual

by

- Thomas Christen (chr@active.ch)
- Jeremy Mawson (jem@loftinspace.com/au (mailto:jem@loftinspace.com.au))

Version 1.1 - 2001/01/28

7.25.1 Table of Contents

- Introduction [p 241]
 - PropertyFile Task [p 241]
 - Entry Task [p 241]
-

7.25.2 Introduction

Ant provides an optional task for editing property files. This is very useful when wanting to make unattended modifications to configuration files for application servers and applications. Currently, the task maintains a working property file with the ability to add properties or make changes to existing ones. However, any comments are lost. Work is being done to make this task a bit more "human friendly".

7.25.3 PropertyFile Task

7.25.3.1 Parameters

Attribute	Description	Required
file	Location of the property file to be edited	Yes
comment	Header for the file itself	no

7.25.3.2 Parameters specified as nested elements

7.25.3.2.1 Entry

Use nested <entry> elements to specify actual modifications to the property file itself

Attribute	Description	Required
key	Name of the property name/value pair	Yes
value	Value to set (=), to add (+) or subtract (-)	Yes
type	Regard the value as : int, date or string (default)	No
operation	"+" or "-" (default) for all datatypes "-" (for date and int only).	No
default	Initial value to set for a property if it is not already defined in the property file. For type date, two additional keywords are allowed: "now" or "never".	No
pattern	For int and date type only. If present, Values will be parsed and formatted accordingly.	No

7.25.3.3 Examples

The following changes the my.properties file. Assume my.properties look like:

```
# A comment
akey=novalue
```

After running, the file would now look like

```
#Thu Nov 02 23:41:47 EST 2000
akey=avalue
adate=2000/11/02 23\:41
anint=1
formatted.int=0014
formatted.date=028 17\:34
```

The slashes conform to the expectations of the Properties class. The file will be stored in a manner so that each character is examined and escaped if necessary. Note that the original comment is now lost. Please keep this in mind when running this task against heavily commented properties files. It may be best to have a commented version in the source tree, copy it to a deployment area, and then run the modifications on the copy. Future versions of PropertyFile will hopefully eliminate this shortcoming.

```
<propertyfile
  file="my.properties"
  comment"My properties" >
  <entry key="akey" value="avalue" />
  <entry key="adate" type="date" value="now"/>
  <entry key="anint" type="int" operation="+"/>
  <entry key="formatted.int" type="int" default="0013" operation="+" pattern="0000"/>
  <entry key="formatted.date" type="date" value="now" pattern="DDD HH:mm"/>
</propertyfile>
```

To produce dates relative from today :

```
<propertyfile
  file="my.properties"
  comment="My properties" >
  <entry key="formatted.date-1"
    type="date" default="now" pattern="DDD"
    operation="-" value="1"/>
  <entry key="formatted.tomorrow"
    type="date" default="now" pattern="DDD"
    operation="+" value="1"/>
</propertyfile>
```

Concatenation of strings :

```
<propertyfile
  file="my.properties"
  comment="My properties" >
  <entry key="progress" default="" operation="+" value="."/>
</propertyfile>
```

Each time called, a "." will be appended to "progress"

7.26 Ant Pvc Task User Manual

by

- Thomas Christensen (tchristensen@nordija.com)
- Don Jeffery (donj@apogeenet.com)

Version 1.1 - 2001/06/27

Problems with UNC pathnames and the use of () in paths are fixed and an updateonly argument introduced.

Version 1.0 - 2001/01/31

Initial release.

7.26.1 Table of Contents

- Introduction [p 244]
 - Pvc Task [p 244]
-

7.26.2 Introduction

The <pvc> task allows the user of ant to extract the latest edition of the source code from a PVCS repository. PVCS is a version control system developed by Merant (<http://www.merant.com/products/pvc>).

Before using this tag, the user running ant must have access to the commands of PVCS (get and pcli) and must have access to the repository. Note that the way to specify the repository is platform dependent so use property to specify location of repository.

This version has been tested against PVCS version 6.5 and 6.6 under Windows and Solaris.

7.26.3 Pvc Task

7.26.3.1 Description

The pvc task is set to point at a PVCS repository and optionally a project within that repository, and can from that specification get the latest version of the files contained by the repository.

7.26.3.2 Parameters

Attribute	Description	Required
repository	The location of the repository (see your PVCS manuals)	Yes
pvcproject	The project within the PVCS repository to extract files from ("/" is root project and that is default if this attribute isn't specified)	No
label	Only files marked with this label are extracted.	No
promotiongroup	Only files within this promotion group are extracted. Using both the <i>label</i> and the <i>promotiongroup</i> tag will cause the files in the promotion group and with that label to be extracted.	No
force	If set to <i>yes</i> all files that exists and are writable are overwritten. Default <i>no</i> causes the files that are writable to be ignored. This stops the PVCS command <i>get</i> to stop asking questions!	No
workspace	By specifying a workspace, the files are extracted to that location. A PVCS workspace is a name for a location of the workfiles and isn't as such the location itself. You define the location for a workspace using the PVCS GUI clients. If this isn't specified the default workspace for the current user is used.	No
pvcsbin	On some systems the PVCS executables <i>pcli</i> and <i>get</i> are not found in the PATH. In such cases this attribute should be set to the bin directory of the PVCS installation containing the executables mentioned before. If this attribute isn't specified the tag expects the executables to be found using the PATH environment variable.	No
ignorereturncode	If set to <i>true</i> the return value from executing the pvc commands are ignored.	No
updateonly	If set to <i>true</i> files are gotten only if newer than existing local files.	No

7.26.3.3 Nested Elements

7.26.3.4 pvcproject element

pvc supports a nested `<pvcproject>` element, that represents a project within the PVCS repository to extract files from. By nesting multiple `<pvcproject>` elements under the `<pvc>` task, multiple projects can be specified.

7.26.3.5 Parameters

Attribute	Description	Required
name	The name of the pvc project	Yes

7.26.3.6 Examples

The following set-up extracts the latest version of the files in the pvcs repository. <!--

```
===== -->
<!-- Get the latest version -->
<!-- ===== -->
<target name="getlatest">
  <pvcS repository="/mnt/pvcs" pvcsproject="/myprj"/>
</target> Now run:

ant getlatest
```

This will cause the following output to appear:

```
getlatest:
[pvcs] PVCS Version Manager (VMGUI) v6.6.10 (Build 870) for Windows NT/80x86
[pvcs] Copyright 1985-2000 MERANT. All rights reserved.
[pvcs] PVCS Version Manager (get) v6.6.10 (Build 870) for Windows NT/80x86
[pvcs] Copyright 1985-2000 MERANT. All rights reserved.
[pvcs] c:\myws\myprj\main.java <- C:\mypvcs\archives\myprj\main.java-arc
[pvcs] rev 1.1
[pvcs] c:\myws\myprj\apache\tool.java <- C:\mypvcs\archives\myprj\apache\tools.java-arc
[pvcs] rev 1.5

BUILD SUCCESSFUL

Total time: 19 seconds
```

This next example extracts the latest version of the files in the pvcs repository from two projects using nested <pvcSproject> elements. <!--

```
===== -->
<!-- Get latest from myprj and myprj2 -->
-->
<!-- ===== -->
<target name="getlatest2">
  <pvcS repository="/mnt/pvcs">
    <pvcSproject name="/myprj" />
    <pvcSproject name="/myprj2" />
  </pvcS>
</target> Now run:

ant getlatest2
```

This will cause the following output to appear:

```
getlatest2:
[pvcs] PVCS Version Manager (VMGUI) v6.6.10 (Build 870) for Windows NT/80x86
[pvcs] Copyright 1985-2000 MERANT. All rights reserved.
[pvcs] PVCS Version Manager (get) v6.6.10 (Build 870) for Windows NT/80x86
[pvcs] Copyright 1985-2000 MERANT. All rights reserved.
[pvcs] c:\myws\myprj\main.java <- C:\mypvcs\archives\myprj\main.java-arc
[pvcs] rev 1.1
[pvcs] c:\myws\myprj\apache\tool.java <- C:\mypvcs\archives\myprj\apache\tool.java-arc
```

```
[pvcS] rev 1.5  
[pvcS] c:\myws\myprj2\apache\tool2.java <- C:\mypvcs\archives\myprj2\apache\tool2.java-arc  
[pvcS] rev 1.2
```

BUILD SUCCESSFUL

Total time: 22 seconds

7.27 RenameExtensions

7.27.1 *Deprecated*

This task has been deprecated. Use the move [p 99] task with a glob mapper [p 29] instead.

7.27.2 *Description*

Renames files in the `srcDir` directory ending with the `fromExtension` string so that they end with the `toExtension` string. Files are only replaced if `replace` is true

See the section on directory based tasks [p 17] , on how the inclusion/exclusion of files works, and how to write patterns. This task forms an implicit FileSet [p 25] and supports all attributes of `<fileset>` (`dir` becomes `srcDir`) as well as the nested `<include>`, `<exclude>` and `<patternset>` elements.

7.27.3 *Parameters*

Attribute	Description	Required
defaultexcludes	indicates whether default excludes should be used or not ("yes"/"no"). Default excludes are used when omitted.	No
excludes	comma separated list of patterns of files that must be excluded. No files (except default excludes) are excluded when omitted.	No
excludesfile	the name of a file. Each line of this file is taken to be an exclude pattern	No
fromExtention	The string that files must end in to be renamed	Yes
includes	comma separated list of patterns of files that must be included. All files are included when omitted.	No
includesfile	the name of a file. Each line of this file is taken to be an include pattern	No
replace	Whether the file being renamed to should be replaced if it already exists	No
srcDir	The starting directory for files to search in	Yes
toExtension	The string that renamed files will end with on completion	Yes

7.27.4 *Examples*

```
<renameext srcDir="/source/project1" includes="*"
excludes="**/samples/*" fromExtension=".java.keep" toExtension=".java" replace="true"/>
```

7.28 Rpm

7.28.1 Description

A basic task for invoking the rpm executable to build a linux installation file. The task currently only works on Linux.

7.28.2 Parameters

Attribute	Description	Required
specFile	The name of the spec File to be used.	Yes
topDir	this is the directory which will have the expected subdirectories, SPECS, SOURCES, BUILD, SRPMS. If this isn't specified, the baseDir value is used	No
cleanBuildDir	this will remove the generated files in the BUILD directory.	No
removeSpec	this will remove the spec file from SPECS	No
command	very similar idea to the cvs task. the default is "-bb"	No
output/error	where standard output and error go	No

7.29 Script

7.29.1 Description

Execute a script in a BSF (<http://oss.software.ibm.com/developerworks/opensource/bsf/>) supported language.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies [p 5] for more information.

All items (tasks, targets, etc) of the running project are accessible from the script, using either their `name` or `id` attributes.

Scripts can do almost anything a task written in Java could do.

7.29.2 Parameters

Attribute	Description	Required
language	The programming language the script is written in. Must be a supported BSF language	No
src	The location of the script as a file, if not inline	No

7.29.3 Examples

```
<project name="squares" default="main" basedir=". ">

  <target name="setup">

    <script language="javascript"> <![CDATA[

      for (i=1; i<=10; i++) {
        echo = squares.createTask("echo");
        main.addTask(echo);
        echo.setMessage(i*i);
      }

    ]]> </script>

  </target>

  <target name="main" depends="setup"/>

</project>
```

generates

```
setup:
```

```
main:
```

```
1
4
9
16
25
36
49
64
81
100
```

```
BUILD SUCCESSFUL
```

Another example, using references by id [p 15] and two different scripting languages:

```
<project name="testscript" default="main">
  <target name="sub">
    <echo id="theEcho"/>
  </target>

  <target name="sub1">
    <script language="netrexx"><![CDATA[
      theEcho.setMessage("In sub1")
      sub.execute
    ]]></script>
  </target>

  <target name="sub2">
    <script language="javascript"><![CDATA[
      theEcho.setMessage("In sub2");
      sub.execute();
    ]]></script>
  </target>

  <target name="main" depends="sub1,sub2"/>
</project>
```

generates

```
sub1:
In sub1
```

```
sub2:
In sub2
```

```
main:
```

```
BUILD SUCCESSFUL
```

7.30 Sound

7.30.1 Description

Plays a sound-file at the end of the build, according to whether the build failed or succeeded. You can specify either a specific sound-file to play, or, if a directory is specified, the `<sound>` task will randomly select a file to play. Note: At this point, the random selection is based on all the files in the directory, not just those ending in appropriate suffixes for sound-files, so be sure you only have sound-files in the directory you specify.

7.30.2 Parameters

Attribute	Description	Required
loops	the number of extra times to play the sound-file; default is 0.	No
duration	the amount of time (in milliseconds) to play the sound-file.	No

To specify the sound-files or the sound-file directories, use the nested `<success>` and `<fail>` elements:

```

<success>    the path to a sound-file directory, or the name of a
              specific sound-file, to be played if the build succeeded.
<fail>       the path to a sound-file directory, or the name of a
              specific sound-file, to be played if the build succeeded.

```

7.30.3 Examples

```

<target name="fun" if="fun" unless="fun.done">
  <sound loops="2">
    <success source="${user.home}/sounds/bell.wav"/>
    <fail source="${user.home}/sounds/ohno.wav"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>

```

plays the `bell.wav` sound-file if the build succeeded, or the `ohno.wav` sound-file if the build failed, three times, if the `fun` property is set to `true`. If the target is a dependency of an "initialization" target that other targets depend on, the `fun.done` property prevents the target from being executed more than once.

```

<target name="fun" if="fun" unless="fun.done">
  <sound>
    <success source="//intranet/sounds/success"/>
    <fail source="//intranet/sounds/failure"/>
  </sound>
  <property name="fun.done" value="true"/>
</target>

```

randomly selects a sound-file to play when the build succeeds or fails.

7.31 Starteam

7.31.1 Description

Checks out files from a StarTeam project.

The *includes* and *excludes* attributes function differently from other tasks in Ant. Multiple patterns must be separated by spaces, not commas. See the examples for more information.

This program makes use of functions from the StarTeam API. As a result this task is only available to licensed users of StarTeam. You must have `starteam-sdk.jar` in your classpath to run this task. For more information about the StarTeam API and how to license it, see the StarBase (<http://www.starbase.com>) web site.

7.31.2 Parameters

Attribute	Description	Required
servername	The name of the StarTeam server.	yes
serverport	The port number of the StarTeam server.	yes
projectname	The name of the StarTeam project.	yes
viewname	The name of the view in the StarTeam project.	yes
username	The username of the account used to log in to the StarTeam server.	yes
password	The password of the account used to log in to the StarTeam server.	yes
targetfolder	The folder to which files are checked out.	yes
foldername	The subfolder in the project from which to check out files.	no
force	Overwrite existing folders if this is set to "true". Defaults to "false".	no
recursion	Indicates if subfolders should be searched for files to check out. Defaults to "true".	no
verbose	Provides progress information. Defaults to "false".	no
includes	Only check out files that match at least one of the patterns in this list. Patterns must be separated by spaces. Patterns in <i>excludes</i> take precedence over patterns in <i>includes</i> .	no
excludes	Do not check out files that match at least one of the patterns in this list. Patterns must be separated by spaces. Patterns in <i>excludes</i> take precedence over patterns in <i>includes</i> .	no

7.31.3 Examples

```
<starteam servername="STARTEAM"
  serverport="49201"
  projectname="AProject"
  viewname="AView"
  username="auser"
  password="secret"
  targetfolder="C:\dev\buildtest\co"
/>
```

The minimum necessary to check out files out from a StarTeam server. This will check out all files in the *AView* view of the *AProject* project to *C:\dev\buildtest\co*.

```
<starteam servername="STARTEAM"
  serverport="49201"
  projectname="AProject"
  viewname="AView"
  username="auser"
  password="secret"
  targetfolder="C:\dev\buildtest\co"
  foldername="\Dev"
  excludes="*.bak *.old"
  force="true"
/>
```

This will checkout all files from the *Dev* folder and below that do not end in *.bak* or *.old*. The force flag will cause any existing files to be overwritten by the version in StarTeam.

```
<starteam servername="STARTEAM"
  serverport="49201"
  projectname="AProject"
  viewname="AView"
  username="auser"
  password="secret"
  targetfolder="C:\dev\buildtest\co"
  includes="*.htm *.html"
  excludes="index.*"
/>
```

This is an example of overlapping *includes* and *excludes* attributes. Because *excludes* takes precedence over *includes*, files named *index.html* will not be checked out by this command.

7.32 Stylebook

7.32.1 Description

This executes the apache Stylebook documentation generator. Unlike the commandline version of this tool, all three arguments are required to run stylebook.

Note: This task depends on external libraries not included in the Ant distribution. See Library Dependencies [p 5] for more information.

7.32.2 Parameters

Attribute	Description	Required
book	the book xml file that the documentation generation starts from	Yes
skindirectory	the directory that contains the stylebook skin	Yes
targetdirectory	the destination directory where the documentation is generated	Yes

The user can also specify the nested `<classpath>` element which defines classpath in which the task is executed.

7.32.3 Examples

```
<stylebook targetdirectory="build/docs"
  book="src/xdocs/book.xml"
  skindirectory="src/skins/myskin"/>
```

The above will generate documentation in build/docs starting from the book src/xdocs/book.xml and using the skin located in directory src/skins/myskin.

7.33 Telnet

7.33.1 Description

Task to automate a remote telnet session. The task uses nested `<read>` to indicate strings to wait for, and `<write>` tags to specify text to send.

If you do specify a userid and password, the system will assume a common unix prompt to wait on. This behavior can be easily over-ridden.

7.33.2 Parameters

Attribute	Values	Required
userid	the login id to use on the telnet server.	No
password	the login password to use on the telnet server.	No
server	the address of the remote telnet server.	Yes
port	the port number of the remote telnet server. Defaults to port 23.	No
initialCR	send a cr after connecting ("yes"). Defaults to "no".	No
timeout	set a default timeout to wait for a response. Specified in seconds. Default is no timeout.	No

7.33.3 Examples

A simple example of connecting to a server and running a command. This assumes a prompt of "ogin:" for the userid, and a prompt of "assword:" for the password.

```
<telnet userid="bob" password="badpass" server="localhost">
  <read>/home/bob</read>
  <write>ls</write>
  <read string="/home/bob"/>
</telnet>
```

This task can be rewritten as:

```
<telnet server="localhost">
  <read>ogin:</read>
  <write>bob</write>
  <read>assword:</read>
  <write>badpass</write>
  <read>/home/bob</read>
  <write>ls</write>
  <read>/home/bob</read>
</telnet>
```

A timeout can be specified at the <telnet> level or at the <read> level. This will connect, issue a sleep command that is suppressed from displaying and wait 10 seconds before quitting.

```
<telnet userid="bob" password="badpass" server="localhost" timeout="20">  
  <read>/home/bob</read>  
  <write echo="false">sleep 15</write>  
  <read timeout="10">/home/bob</read>  
</telnet>
```

The task can be used with other ports as well:

```
<telnet port="80" server="localhost" timeout="20">  
  <read/>  
  <write>GET / http/0.9</write>  
  <write/>  
  <read timeout="10">&lt;/HTML&gt;</read>  
</telnet>
```

7.34 Test

7.34.1 Description

This is a primitive task to execute a unit test in the org.apache.testlet framework.

7.34.2 Parameters

Attribute	Description	Required
showSuccess	a boolean value indicating whether tests should display a message on success	No
showBanner	a boolean value indicating whether a banner should be displayed when starting testlet engine	No
forceShowTrace	a boolean indicating that a stack trace is displayed on any failure	No
showTrace	a boolean indicating that a stack trace is displayed on error (but not normal failure)	No

The user can also specify the nested <classpath> element which defines classpath in which the task is executed. The user also specifies a subelement per testlet executed which has content that specifies tasklet classname.

7.34.3 Examples

```
<test showSuccess="false" showBanner="false" showTrace="true" forceShowTrace="true">
  <classpath refid="test.classpath"/>
  <testlet>org.foo.MyTestlet</testlet>
  <testlet>org.foo.MyOtherTestlet</testlet>
</test>
```

The above will run the testlets org.foo.MyTestlet and org.foo.MyOtherTestlet

7.35 wljspc

7.35.0.1 Description

Class to precompile JSP's using weblogic's jsp compiler (weblogic.jspc)

Tested only on Weblogic 4.5.1 - NT4.0 and Solaris 5.7,5.8

7.35.0.2 Parameters

Attribute	Values	Required
src	oot of source tree for JSP, ie, the document root for your weblogic server	Yes
dest	root of destination directory, what you have set as WorkingDir in the weblogic properties	Yes
package	start package name under which your JSP's would be compiled	Yes
classpath	Class path to use when compiling jsp's	Yes

A classpath should be set which contains the weblogic classes as well as all application classes referenced by the JSP. The system classpath is also appended when the jspc is called, so you may choose to put everything in the classpath while calling Ant. However, since presumably the JSP's will reference

classes being build by Ant, it would be better to explicitly add the classpath in the task

The task checks timestamps on the JSP's and the generated classes, and compiles

only those files that have changed.

It follows the weblogic naming convention of putting classes in

`_dirName/_fileName.class` for `dirname/fileName.jsp`

7.35.0.3

Example

```
<target name="jspcompile" depends="compile">
<wljspc src="c:\\weblogic\\myserver\\public_html" dest="c:\\weblogic\\myserver\\serverclasses"
package="myapp.jsp" >
<classpath>
<pathelement location="${weblogic.classpath}" />
<pathelement path="${compile.dest}" />
</classpath>
</wljspc>
</target>
```

7.35.0.4

Limitations

This works only on weblogic 4.5.1

It compiles the files thru the Classic compiler only.

Since it is my experience that weblogic jspc throws out of memory error on being given too many files at one go, it is called multiple times with one jsp file each.

7.36 XMLValidate

7.36.1 Description

This task checks xml files are valid (or only well formed). The task uses crimson SAX2 parser implementation by default, but one can specify any SAX1/2 parser if needed

7.36.2 Parameters

Attribute	Description	Required
file	the file(s) you want to check. (optionally can use an embedded fileset)	No
lenient	if true, only check the xml document is well formed (ignored if the specified parser is as SAX1 parser)	No
classname	the parser to use. (default: crimson).	No
classpathref	where to find the parser class. Optionally can use an embedded classpath element.	No
failonerror	fails on a error if set to true (defaults to true).	No
warn	log parser warn events.	No

7.36.3 Examples

```
<xmlvalidate file="toto.xml"/>
```

```
<xmlvalidate failonerror="no" lenient="yes" warn="yes"
  classname="org.apache.xerces.parsers.SAXParser" >
  classpath="lib/xerces.jar">
  <fileset dir="src" includes="style/*.xsl"/>
</xmlvalidate>
```

8 IDE Integration

8.1 About Antidote

Version 0.1 (2001/02/13)

Authors: Simeon H.K. Fitch (mailto:simeon@fitch.net)

8.1.1 Overview

Antidote is the Ant (<http://jakarta.apache.org/ant/index.html>) subproject for developing a graphical user interface to facilitate the efficient use of Ant. In general, its purpose is to allow the quick generation, modification, and use of Ant build files, helping the user define a build process and track down build problems. It is not meant to be an IDE, but an enabler for the powerful features available in Ant, particularly for novice users, or users who want a rapid way of controlling their build process.

8.1.2 Status

Antidote is still in the early stages of development, but does have a set of usable features, including:

- Reading Ant build files.
- Selecting targets and executing them.
- Context highlighted build status console.
- Modification of (some) build file components.
- Saving modified build file.

Current development tasks include:

- A more complete set of target and task editing capabilities.
- A wizard for creating basic build files, including importing existing code bases.
- Better build progress monitoring.

The Antidote source distribution comes with requirements and design documentation that better cover the details of application architecture, how to develop against it, and what the long term goals are. Furthermore, there is a TODO file listing the detailed, near-term tasks that need accomplishing.

8.1.3 Getting Involved

The source code for Antidote is located in a separate Module (`jakarta-ant-antidote` (<http://cvs.apache.org/viewcvs/jakarta-ant-antidote/>)) in CVS. All the existing documentation can be found there where new contributors should read:

- Design Overview (<http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/developer/design/design-overview.html>)
- Feature List (<http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/develop->

[oper/design/gui-requirements.html](http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/developer/design/gui-requirements.html))

- Idea Refinement (<http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/developer/design/gui-ideas.txt>)
- New Module HOWTO (<http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/developer/design/new-module-howto.html>)
- Static Class Diagrams (<http://cvs.apache.org/viewcvs/~checkout~/jakarta-ant-antidote/docs/developer/design/uml/index.html>)

Online discussions about Antidote occur on the jakarta-ant mailing list (<http://jakarta.apache.org/site/mail.html>). The application infrastructure is fairly complete, but there are almost unlimited opportunities for feature contributions.

Aspiring contributors new to the Jakarta Project should (carefully) read the following for details on the contribution process:

- Get Involved (<http://jakarta.apache.org/site/getinvolved.html>)
- Project Guidelines (<http://jakarta.apache.org/site/guidelines.html>)
- Source Repositories (how to contribute patches) (<http://jakarta.apache.org/site/source.html>)

8.2 AntRunner For JBuilder (unbundled)

See <http://www.dieter-bogdoll.de/java/AntRunner/>.

8.3 AntWork Plugin for the Jext - Java Text Editor

by

- Klaus Hartlage (KHartlage@t-online.de)

Version \$Revision: 1.1 \$ - \$Date: 2001/03/05 13:07:15 \$

You can download the plugin at: <ftp://jext.sourceforge.net/pub/jext/plugins/AntWork.zip>

8.3.1 *Installation instructions from the Readme.txt:*

You have to enable the Jext Console to see the Ant output (menu: Edit->Options... - General Panel), because the Ant messages are redirected to the Jext console.

You can configure the Ant call in the Jext menu: Edit->Options... - Plugin Options - Antwork Plugin Panel; here you can set the ant home directory and the path to your build file.

You can start AntWork in the menu: Plugins->Ant->Work Now! In the appearing dialog box you can enter the target which you want to compile.

If a javac error occurs in the ant run an error-list opens within Jext. With a double-click on the error-message you jump to the error in the specified java text file.

8.4 NetBeans Forte for Java (unbundled)

See <http://ant.netbeans.org/>.

8.5 Visual Age for Java Tasks and Plugin User Manual

by

- Wolf Siberski (wolf.siberski@tui.de)
- Christoph Wilhelms (christoph.wilhelms@tui.de)

Version 1.1 - 2001/02/14

8.5.1 *Table of Contents*

- Introduction [p 266]
 - The Tasks [p 267] [p 266]
 - VAJLoad [p 267]
 - VAJExport [p 267]
 - VAJImport [p 269]
 - A sample build file [p 270]
 - The Plugin [p 271]
 - Installation [p 271]
 - Usage [p 272]
 - Frequently Asked Questions [p 273]
 - Visual Age Versions [p 274]
 - History [p 275] [p 275]
-

8.5.2 *Introduction*

Visual Age for Java is a great Java IDE, but it lacks decent build support; for creating deliveries. On the other hand, Ant supports the build process very good, but is (at least at the moment) command line based. So we decided to write some tasks to access the VAJ repository and a small visual Ant frontend to make running Ant from VAJ possible. We use the Tool API to integrate Ant in VisualAge for Java. In combination with the VAJ tasks (vajload, vajexport, vajimport) you can load defined defined versions of projects into your workspace, export the source code, compile it with an external compiler and build a jar without leaving the IDE. Of course compile messages are viewed in a logging window. Concluding: This tool provides decent deployment support VAJ has not (out of the box).

8.5.3 The Tasks

At the moment there are three tasks which help integrating the VAJ repository contents into an external build process:

VAJLoad [p 267]	loads specified versions into the workspace
VAJExport [p 267]	exports specified packages into the file system
VAJImport [p 269]	imports specified files into the workspace

These tasks are described in detail below.

8.5.4 VAJLoad

8.5.4.1 Description:

Loads a specified VAJ project version into the workspace.

8.5.4.2 Parameters

none

8.5.4.3 Parameters specified as nested elements

8.5.4.3.1 vajproject

Attribute	Description	Required
name	name of the VAJ project to load into the workspace	yes
version	name of the requested version	yes

8.5.4.4 Example

```
<vajload>  
  <vajproject name="My Testcases" version="1.7beta" />  
  <vajproject name="JUnit" version="3.2" />  
</vajload>
```

8.5.5 VAJExport

8.5.5.1 Description:

Exports Java source files, class files and/or resources from the workspace to the file system. Exports can be specified by giving the VAJ project name and package name(s). This works very similar to FileSets [p 25] .

8.5.5.2 Parameters

Attribute	Description	Required
destdir	location to store the exported files	yes
exportSources	export source files (default: "yes")	no
exportResources	export resource files (default: "yes")	no
exportClasses	export class files (default: "no")	no
exportDebugInfo	include debug info in exported class files (default: "no")	no
defaultexcludes	use default excludes when exporting (default: "yes")	no

8.5.5.3 Parameters specified as nested elements

8.5.5.3.1 *include*

specifies the packages to include into the export

Attribute	Description	Required
name	name of the VAJ project and package to export. The first element of the name must be the project name, then the package name elements separated by '/'. 	yes

8.5.5.3.2 *exclude*

specifies the packages to exclude from the export

Attribute	Description	Required
name	name of the VAJ project/package not to export	yes

8.5.5.4 Example

```
<vajexport destdir="${src.dir}" exportResources="no">
  <include name="MyProject/**"/>
  <exclude name="MyProject/test/**"/>
</vajexport>
```

This example exports all packages in the VAJ project 'MyProject', except packages starting with 'test'.

8.5.5.5 Default Excludes

The default excludes are:

```
IBM**/**
Java class libraries/**
Sun class libraries/**
JSP Page Compile Generated Code/**
VisualAge**/**
```

8.5.6 VAJImport

8.5.6.1 Description:

Imports Java source files, class files and/or resources from the file system into VAJ. These imports can be specified with a fileset.

8.5.6.2 Parameters

Attribute	Description	Required
vajProject	imported files are added to this VAJ project	yes
importSources	export source files (default: "yes")	no
importResources	export resource files (default: "yes")	no
importClasses	export class files (default: "no")	no

8.5.6.3 Parameters specified as nested elements

8.5.6.3.1 *fileset*

A FileSet [p 25] specifies the files to import.

8.5.6.4 Example

```
<vajimport project="Test" importClasses="true">
  <fileset dir="${import.dir}">
    <include name="com/sample/**/* .class"/>
    <exclude name="com/sample/test/**"/>
  </fileset>
</vajimport>
```

This example imports all class files in the directory `${import.dir}/com/sample` excluding those in the subdirectory `test`

8.5.7 A sample build file

This is a sample build file which builds a zip file useful for distribution of the VAJ plugin. The build file exports a specific version of Ant, compiles it, extracts necessary helper classes and zips everything together. This is useful as complete example as well as to deploy Ant in a team where all developers use VAJ. The resulting zip file must be unzipped in <VAJIn-stallDir>\ide\tools\org-apache-tools-ant. For more information see the installation [p 271] section.

```
<?xml version="1.0"?>
<!-- ===== -->
<!-- Builds a binary distribution of the VAJ Ant Plugin -->
<!-- ===== -->
<project name="anttool" default="buildall" basedir="c:\temp\anttool">

  <property name="src.dir" value="${basedir}/src"/>
  <property name="zip.dir" value="${basedir}/dist"/>
  <property name="build.classes" value="${basedir}/classes"/>

  <!-- location of the unzipped Ant source distribution -->
  <property name="antdistribution.dir" value="u:\ant-1.3beta1"/>

  <!-- VAJ install dir -->
  <property name="vaj.dir" value="C:\IBMJava2"/>

  <path id="classpath">
    <pathelement location="${src.dir}"/>
    <pathelement location="${vaj.dir}\IDE\project_resources\IBM IDE Utility class libraries" />
    <pathelement location="${build.classes}"/>
  </path>

  <!-- ===== -->
  <!-- Load Projects into Workspace -->
  <!-- ===== -->
  <target name="load" description="load projects">
    <vajload>
      <project name="Ant" version="1.3"/>
      <project name="Apache Oro RegExp" version="2.0.1"/>
    </vajload>
  </target>

  <!-- ===== -->
  <!-- Creates the build dirs -->
  <!-- ===== -->
  <target name="preparedirs">
    <mkdir dir="${basedir}"/>
    <mkdir dir="${src.dir}"/>
    <mkdir dir="${zip.dir}"/>
    <mkdir dir="${build.classes}"/>
  </target>

  <!-- ===== -->
  <!-- Export from VA -->
  <!-- ===== -->
  <target name="export" depends="preparedirs" description="export projects">
    <mkdir dir="${src.dir}"/>
    <!-- Export all Ant sources except optional tasks and the Oro sources -->
    <vajexport destdir="${src.dir}">
      <include name="Ant/**"/>
      <exclude name="Ant/**/optional/**"/>
      <exclude name="Ant/org/apache/tools/ant/gui/**"/>
      <include name="Apache Oro/**"/>
    </vajexport>
    <!-- Export selected optional tasks -->
    <vajexport destdir="${src.dir}">
      <include name="Ant/org/apache/tools/ant/taskdefs/optional/ide/**"/>
    </vajexport>
  </target>

  <!-- ===== -->
  <!-- Unzip necessary libraries -->
  <!-- ===== -->
  <target name="expand">
    <mkdir dir="${build.classes}"/>
    <unzip src="${antdistribution.dir}\lib\crimson.jar"
      dest="${build.classes}"
    />
    <unzip src="${antdistribution.dir}\lib\jaxp.jar"
      dest="${build.classes}"
    />
    <delete dir="${build.classes}\META-INF" />
  </target>

  <!-- ===== -->
```

```

<!-- Compile the source code -->
<!-- ===== -->
<target name="compile" depends="expand" description="compile java sources">
  <mkdir dir="${build.classes}"/>
  <javac srcdir="${src.dir}"
    destdir="${build.classes}"
    debug="on"
    deprecation="off"
    optimize="on" >
    <classpath refid="classpath" />
    <include name="org/apache/**"/>
    <exclude name="**/JakartaRegexpMatcher.java"/>
  </javac>
</target>

<!-- ===== -->
<!-- Copy resources and zip everything together -->
<!-- ===== -->
<target name="zip" depends="compile" description="creates zip distribution">
  <copy todir="${build.classes}">
    <fileset dir="${src.dir}">
      <include name="org/apache/**"/>
      <exclude name="**/*.java"/>
    </fileset>
  </copy>
  <mkdir dir="${build.classes}/doc"/>
  <copy todir="${build.classes}/doc" overwrite="yes">
    <fileset dir="${antdistribution.dir}/docs">
      <include name="VAJAnttool.html, toolmenu.gif, anttool1.gif"/>
    </fileset>
  </copy>
  <copy file="${antdistribution.dir}/src/main/org/apache/tools/ant/taskdefs/optional/ide/default.ini" todir="${build.classes}" overwrite="yes"/>
  <mkdir dir="${zip.dir}"/>
  <zip basedir="${build.classes}" zipfile="${zip.dir}/anttool.zip" />
</target>

<!-- ===== -->
<!-- Cleans source and class dirs -->
<!-- ===== -->
<target name="clean" depends="preparedirs" description="removes all files from src and build tree">
  <delete>
    <fileset dir="${src.dir}"/>
  </delete>
  <delete>
    <fileset dir="${build.classes}"/>
  </delete>
  <delete>
    <fileset dir="${zip.dir}"/>
  </delete>
</target>

<!-- ===== -->
<!-- Cleans the build dir, loads required project versions, exports, -->
<!-- compiles and zips the Plugin -->
<!-- ===== -->
<target name="buildall" depends="clean, load, export, zip" description="build all">
</target>
</project>

```

8.5.8 The Plugin

The tasks are usable within VAJ by running the `org.apache.tools.ant.Main` class, but this is quite inconvenient. Therefore a small GUI is provided which allows selecting a build file and executing its targets. This Plugin is accessible from the VAJ Tools menu (see Usage [p 272]).

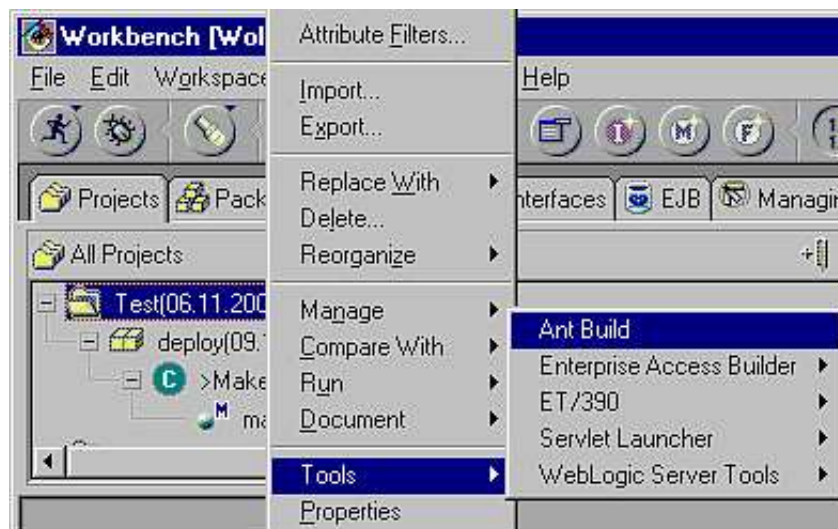
8.5.9 Installation

At the moment the installation has it's rough edges. If something described below doesn't work for You, it's probably not Your fault but incomplete/wrong instructions. In this case, please contact one of the authors [p 266] . We assume `C:\IBMVJava` as VAJ install directory. If You have installed it elsewhere, adapt the pathes below. Execute following steps to get the PlugIn up and running:

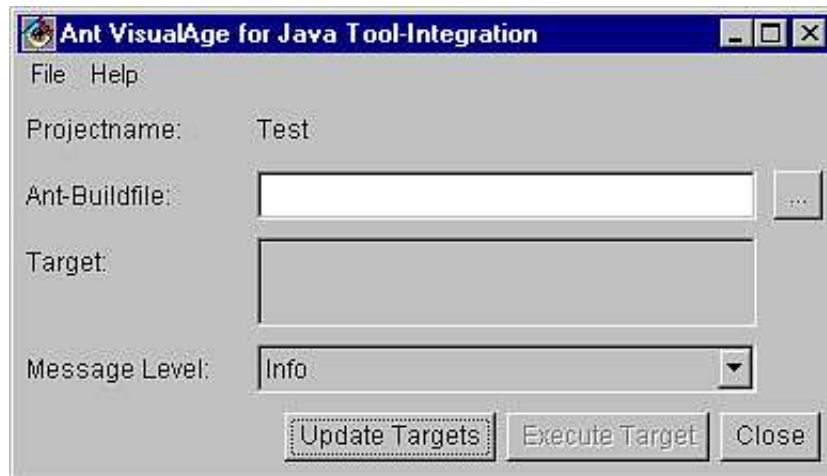
- install the Visual Age IDE Tools (via File->Quick Start-> Add feature->'IBM IDE Utility class libraries')
- import an appropriate XML parser to VAJ (we use Xerces 1.2.0 and are happy with it). Unfortunately the XML parser delivered with VAJ (in the project 'IBM XML Parser for Java') doesn't work with Ant. You have to remove that project (temporarily) from the workspace before importing another XML implementation.
- import the Ant sources into VAJ.
- Create the directory C:\IBMJava\ide\tools\org-apache-tools-ant.
- export the Ant and XML parser class files into this directory. Be sure to select class files and resources. Sources don't have to be exported. Some optional tasks have errors and can't be exported when You don't have the necessary packages in Your workspace (e.g. junit task, ejbc task). If You need this tasks either import these packages into VAJ, too, or copy the .class files directly from the binary distribution.
- copy default.ini (in jakarta-ant\src\...\taskdefs\optional\ide) to C:\IBMJava\ide\tools\org-apache-tools-ant\default.ini.
- if you want to access this help from the Workbench, create the directory C:\IBMJava\ide\tools\org-apache-tools-ant\doc and copy the files VAJAntTool.html, toolmenu.gif and anttool1.gif to it.
- VAJ has to be restarted to recognize the new tool.
- Now if You open the context menu of a project, You should see the entry 'Ant Build' in the Tools submenu (see Usage [p 272]).
- Make sure the tool works as expected. Now You can remove Ant and the imported XML parser from Your workspace (and optionally add the IBM parser again).

8.5.10 Usage

Being sure the tool is installed correctly and your Ant build file is configured, it is really easy to use. Go to your Workbench, select the project you want to deploy and open its context menu. In the submenu *Tools* you should find the new entry *Ant Build*. Click it to start the tool!



After a short time this frame should pop up:



This frame contains the following elements:

- A menubar with some options described later
- The name of your selected VAJ project
- An entry field for the Ant XML buildfile with a browse [...] button. The full qualified filename, including the directory is needed here.
- A list with tasks specified in the buildfile. Until your first save of the build info (described later), this list will be empty. When loading a build file by the *Reload Project* button, this list is filled with all tasks which have a description attribute. The task you select in this list will be executed when pressing the *Execute Task* button.
- A pulldown box for specifying the log level.
- Three buttons. Two of them I have already described and the third one is just the *Close* button to exit our small tool!

After you have set up your buildprocess you might find it useful to save the data you've just entered, so we implemented an option to save it to the repository into your selected project. Make sure that you have an open edition of your project before selecting *Save BuildInfo To Repository* from the *File* menu. Now your information is saved to this edition of your project and will be loaded automatically the next time you start *Ant Build*.

If you have closed the log window accidentally, it can be reopened with the *Log* item in the *File* menu, and if you want to know who developed this, just select *About* in the *Help* menu.

8.5.11 Frequently Asked Questions

Q: I want to load, export and build more then one Visual Age project to one jar! How to?

A: The VA tasks are able to load and export several Projects all at once. You can choose whatever project you like for storing the tool information, it doesn't really matter

Q: When I load my build file, the list of targets is empty. Why?

A: You need to add the optional "description" parameter to the targets You want to come up in the list. Then reload the build file in the "ant build" tool. We chose to display only targets with description to allow the build file developer to distinguish between targets for end users and helper targets.

Q: Is there a sample build file available?

A: Now You can find an example [p 270] in this manual

Q: Why does it export my entire workspace when I've already implicitly selected a project when starting the Tool?

A: This selection does not carry into the buildfile you are using. Set the Project name at the beginning of the "includes" parameter.

Q: When I import Ant into my Workspace, I get Problems reported. Can I ignore them?

A: It depends on the problems reported, and what you want to do with Ant. Problems you can't ignore:

- Classes from javax.xml.parser missing - install a compatible parser (see installation [p 271])
- Classes from com.ibm.ivj.util missing - install the Visual Age IDE Utility feature (see installation [p 271]).

Q: Why is the task name vajload when the class is called VAJLoadProjects?

A: At the moment this task can load only project versions. This is reflected by the class name. It may be extended to load package and class versions, too, and we didn't want to let these extensions affect the build files. Therefore we chose the more general name vajload as task name.

Q: I want to use the same buildfile both within Visual Age and from the command line using my regular Ant environment. What do I need to be aware of?

A: The three Visual Age Tasks won't work when executing Ant from the command line.

Q: How do I control the import/export of sourcefiles, compiled files and project resources explicitly?

A: Via the Boolean values exportClasses (default false) exportSources (default true) and exportResources (default true). In some situations, Resources are not exported correctly without this being explicitly set.

8.5.12 Known Problems

- Exporting a package containing just resources doesn't work. This is a VAJ Tool API bug. Workaround: create a dummy class and set 'exportSources' to false.

8.5.13 VisualAge for Java Versions

This tool integration has been tested with versions 3.02 and 3.5 of VisualAge for Java. It should run with the 2.x Versions, too, but we didn't try. The graphical user interface is build with AWT so it is JDK independent by now.

8.5.14 History

1.0	2000/09/11	Initial Version
1.1	2001/02/14	Added Task documentation and more FAQs (thanks to Richard Bourke for the FAQ additions)

9 Developing with Ant

9.1 Writing Your Own Task

It is very easy to write your own task:

1. Create a Java class that extends `org.apache.tools.ant.Task`.
2. For each attribute, write a *setter* method. The setter method must be a `public void` method that takes a single argument. The name of the method must begin with `set`, followed by the attribute name, with the first character of the name in uppercase, and the rest in lowercase. The type of the attribute can be:
 - `String`
 - any primitive type (they are converted for you from their `String`-representation in the buildfile)
 - `boolean` - your method will be passed the value `true` if the value specified in the buildfile is one of `true`, `yes`, or `on`)
 - `Class`
 - `File` (in which case the value of the attribute is interpreted relative to the project's basedir)
 - any other type that has a constructor with a single `String` argument
3. If your task has enumerated attributes, you should consider using a subclass of `org.apache.tools.ant.types.EnumeratedAttribute` as an argument to your setter method. See `org/apache/tools/ant/taskdefs/FixCRLF.java` for an example.
4. If the task should support character data, write a `public void addText(String)` method.
5. For each nested element, write a *create* or *add* method. A create method must be a `public` method that takes no arguments and returns an `Object` type. The name of the create method must begin with `create`, followed by the element name. An add method must be a `public void` method that takes a single argument of an `Object` type with a no-argument constructor. The name of the add method must begin with `add`, followed by the element name.
6. Write a `public void execute` method, with no arguments, that throws a `BuildException`. This method implements the task itself.

9.1.1 The Life-cycle of a Task

1. The task gets instantiated using a no-argument constructor, at parser time. This means even tasks that are never executed get instantiated.
2. The task gets references to its project and location inside the buildfile via its inherited `project` and `location` variables.
3. If the user specified an `id` attribute to this task, the project registers a reference to this newly created task, at parser time.
4. The task gets a reference to the target it belongs to via its inherited `target` variable.
5. `init()` is called at parser time.
6. All child elements of the XML element corresponding to this task are created via this task's `createXXX()` methods or instantiated and added to this task via its `addXXX()` methods, at parser time.
7. All attributes of this task get set via their corresponding `setXXX` methods, at runtime.
8. The content character data sections inside the XML element corresponding to this task is added to the

task via its `addText` method, at runtime.

9. All attributes of all child elements get set via their corresponding `setXXX` methods, at runtime.
10. `execute()` is called at runtime. While the above initialization steps only occur once, the `execute()` method may be called more than once, if the task is invoked more than once. For example, if `target1` and `target2` both depend on `target3`, then running `'ant target1 target2'` will run all tasks in `target3` twice.

9.1.2 Example

Let's write our own task, which prints a message on the `System.out` stream. The task has one attribute, called `message`.

```
package com.mydomain;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class MyVeryOwnTask extends Task {
    private String msg;

    // The method executing the task
    public void execute() throws BuildException {
        System.out.println(msg);
    }

    // The setter for the "message" attribute
    public void setMessage(String msg) {
        this.msg = msg;
    }
}
```

It's really this simple ;-)

Adding your task to the system is rather simple too:

1. Make sure the class that implements your task is in the classpath when starting Ant.
2. Add a `<taskdef>` element to your project. This actually adds your task to the system.
3. Use your task in the rest of the buildfile.

9.1.3 Example

```
<?xml version="1.0"?>

<project name="OwnTaskExample" default="main" basedir=".">
    <taskdef name="mytask" classname="com.mydomain.MyVeryOwnTask"/>

    <target name="main">
        <mytask message="Hello World! MyVeryOwnTask works!"/>
    </target>
</project>
```

Another way to add a task (more permanently), is to add the task name and implementing class name to the `default.properties` file in the `org.apache.tools.ant.taskdefs` package. Then you can use it as if it were a built-in task.

9.2 Build Events

Ant is capable of generating build events as it performs the tasks necessary to build a project. Listeners can be attached to Ant to receive these events. This capability could be used, for example, to connect Ant to a GUI or to integrate Ant with an IDE.

To use build events you need to create an ant `Project` object. You can then call the `addBuildListener` method to add your listener to the project. Your listener must implement the `org.apache.tools.ant.BuildListener` interface. The listener will receive `BuildEvents` for the following events

- Build started
- Build finished
- Target started
- Target finished
- Task started
- Task finished
- Message logged

If you wish to attach a listener from the command line you may use the `-listener` option. For example:

```
ant -listener org.apache.tools.ant.XmlLogger
```

will run Ant with a listener that generates an XML representation of the build progress. This listener is included with Ant, as is the default listener, which generates the logging to standard output.

10 Ant API

See <http://jakarta.apache.org/ant/manual/api/index.html>.

11 License

```

/*
 * =====
 *                               The Apache Software License, Version 1.1
 * =====
 *
 * Copyright (C) 1999 The Apache Software Foundation. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modifica-
 * tion, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The end-user documentation included with the redistribution, if any, must
 *    include the following acknowledgment: "This product includes software
 *    developed by the Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself, if
 *    and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Ant" and "Apache Software Foundation" must not be used to
 *    endorse or promote products derived from this software without prior
 *    written permission. For written permission, please contact
 *    apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache", nor may
 *    "Apache" appear in their name, without prior written permission of the
 *    Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
 * FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
 * INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLU-
 * DING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * This software consists of voluntary contributions made by many individuals
 * on behalf of the Apache Software Foundation. For more information on the
 * Apache Software Foundation, please see <http://www.apache.org/>.
 */

```

12 Feedback

To provide feedback on this software, please subscribe to the Ant User Mailing List (<http://jakarta.apache.org/site/mail.html>)

If you want to contribute to Ant or stay current with the latest development, join the Ant Development Mailing List (<http://jakarta.apache.org/site/mail.html>)

Archives of both lists can be found at <http://archive.covalent.net/>. Many thanks to Covalent Technologies. A searchable archive can be found at <http://marc.theaimsgroup.com>. If you know of any additional archive sites, please report them to the lists.